Basis software en een GKS Driver voor een grafische kaart in een UNIX systeem

van Beek, D.A.

Award date:
1985

AFDELING DER ELEKTROTECHNIEK
TECHNISCHE HOGESCHOOL EINDHOVEN
Vakgroep Digitale Systemen

Afstudeerverslag

BASIS SOFTWARE EN EEN GKS DRIVER VOOR EEN GRAFISCHE KAART IN EEN UNIX SYSTEEM

D.A. van Beek

Hoogleraar: Prof. Ir. A. Heetman

Mentor: Ing. L.A. van Bokhoven / Ing. P.H.A. van de Putten

Periode: 1-5-1984 tot 1-12-1985 (part-time)

Inhoudsopgave blz

Bijlagen

1 Samenvatting

In het kader van het THE-KUNix project werken de TH Eindhoven en de Katholieke Universiteit Nijmegen samen aan de ontwikkeling van een Unix machine. Op dit Unix systeem moet uiteindelijk de nieuwe grafische standaard GKS geimplementeerd worden.
Het uitgangspunt bij mijn afstudeeropdracht was een nieuwe, op de vakgroep Digitale Systemen ontwikkelde, grafische kaart. Voor deze grafische kaart is basissoftware geschreven, met behulp waarvan alle hardware-technisch realiseerbare functies van de grafische kaart softwarematig ondersteund worden. De grafische kaart met de bijbehorende basissoftware is momenteel operationeel in een stand-alone M68000 systeem.
Voor implementatie van de grafische kaart in het Unix systeem, zijn er twee software modules geschreven. De ene module kan onder Unix de grafische kaart aansturen, zodanig dat er een stand alone grafisch systeem ontstaat, analoog als in het M68000 systeem. Met behulp van de tweede module kan GKS geimplementeerd worden. Hiervoor moest, naast de basissoftware die onder Unix de grafische kaart aanstuurt, tevens een GKS driver geschreven worden. De driver is een laag software die de interface verzorgt tussen GKS en het grafische systeem.
De ontwikkelde software is logisch en gestructureerd van opbouw. De sources hebben een overzichtelijke lay out. Ook is aandacht geschonken aan een grote mate van portability (overdraagbaarheid) van de software. Bij de software ontwikkeling is tevens gebruik gemaakt van de kennis van programmeertechnieken, die is opgedaan bij de bestudering van de GKS sources. Bij de bestudering van de GKS sources werd ook inzicht verkregen in de interne werking van GKS. Deze interne werking van GKS en wel speciaal in relatie tot het aansturen van de driver, is in dit verslag beschreven.
Om de software ontwikkeling zoveel mogelijk te automatiseren en ondersteunen, zijn er verder nog een aantal utilities ontwikkeld. De belangrijkste van deze utilities zijn voor algemeen gebruik beschikbaar onder Unix.
Ten behoeve van de voortzetting van dit werk door volgende afstudeerders, is het ontwerp aangegeven van het toekomstige grafische systeem, waarop GKS onder Unix geimplementeerd zal moeten worden. Tevens is een overzicht gegeven van alle nu nog te verrichten werkzaamheden in het kader van het grafische project.

Hierbij wil ik gaarne nog een woord van dank uitspreken aan mijn coach ing. L.A. van Bokhoven, die helaas net voor het einde van mijn afstuderen de vakgroep heeft verlaten. Ik wil hem danken voor de zeer plezierige wijze van samenwerking, voor zijn waardevolle adviezen en vooral ook voor de grote mate van zelfstandigheid die hij mij gelaten heeft. Ing. P.H.A. van de Putten, die na het vertrek van de heer van

Bokhoven mijn coaching heeft overgenomen, wil ik danken voor het feit dat hij ondanks zijn drukke werkzaamheden nog tijd heeft kunnen vinden om mij te begeleiden. Ik denk hierbij in het bijzonder aan zijn waardevolle adviezen in betrekking tot mijn tussentijdse voordracht en mijn afstudeervoordracht.
Verder wil ik de gehele vakgroep Digitale Systemen danken, voor de prettige werksfeer en de altijd aanwezige bereidheid tot het verlenen van hulp. In het bijzonder wil ik hierbij nog noemen Ing. C.H. van Hooidonk en Ing. C.F.J.M. Stork voor hun praktische hulp bij de uitvoering van mijn afstudeerwerk.

2 Afkortingen

C.W.I.	Centrum voor Wiskunde en Informatica te Amsterdam.
DI/DD	Device Independent/Device Dependent. Dit is een vorm van een interface.
EB	Naam van de vakgroep Digitale Systemen, afdeling Elektrotechniek van de THE.
GDC	Grafical Display Controller: IC t.b.v. grafische toepassingen.
geb	Grafisch werkstation, ontwikkeld op de vakgroep EB (Digitale Systemen), afdeling Elektrotechniek van de THE.
GEB001	Naam van het grafisch werkstation dat ontwikkeld is op de vakgroep EB van de THE. 001 is het volgnummer van het werkstation.
GKS	The Grafical Kernel System: Standaard voor grafische toepassingen op computers.
IO	Input Output
RAM	Random Access Memory
THE-KUNix	Samenwerkingsverband tussen de vakgroep Digitale Systemen, afdeling Elektrotechniek van de T.H. Eindhoven en de faculteit Wiskunde en Natuurwetenschappen van de Katholieke Universiteit Nijmegen. Het doel van dit samenwerkingsverband is het ontwikkelen van een eigen Unix systeem.
tty	teletype. Onder Unix vaak gebruikt als aanduiding van een Unix (beeldscherm-) terminal.

3 Inleiding

Reeds geruime tijd werken de Technische Hogeschool Eindhoven en de Katholieke Universiteit Nijmegen samen aan het THE-KUNix project. Dit project omvat de ontwikkeling en bouw van een compleet UNIX systeem. Een onderdeel van dit systeem is een grafische kaart, die ontwikkeld is op de vakgroep Digitale Systemen, afdeling Elektrotechniek van de TH Eindhoven. Met behulp van deze kaart en de nodige basis software zou uiteindelijk de nieuwe grafische standaard GKS geimplementeerd moeten worden.
GKS, the Grafical Kernel System, is een pakket bestaande uit een aantal gestandaardiseerde grafische procedures. Tijdens mijn afstuderen heb ik de benodigde basissoftware voor de grafische kaart geschreven en uitgetest. Tevens is er in GKS een driver geschreven voor de grafische kaart, d.w.z. een stuk software dat de koppeling verzorgt tussen GKS en de grafische kaart.
Bij de ontwikkeling van de software is ook aandacht besteed aan het tot stand brengen van een goede software ontwikkelomgeving. D.w.z. het creeren van software, die het ontwikkelen van de software voor de grafische kaart zo veel mogelijk automatiseert en controleert. Verder is er naar gestreefd de software zo logisch, structureel en doorzichtig mogelijk op te stellen. De software is bovendien zodanig voorzien van commentaar, dat er een betrouwbaar en inzichtelijk geheel wordt verkregen.

4 Globaal Overzicht van de Grafische Systemen

4.1 Globale Werking Grafische Kaart

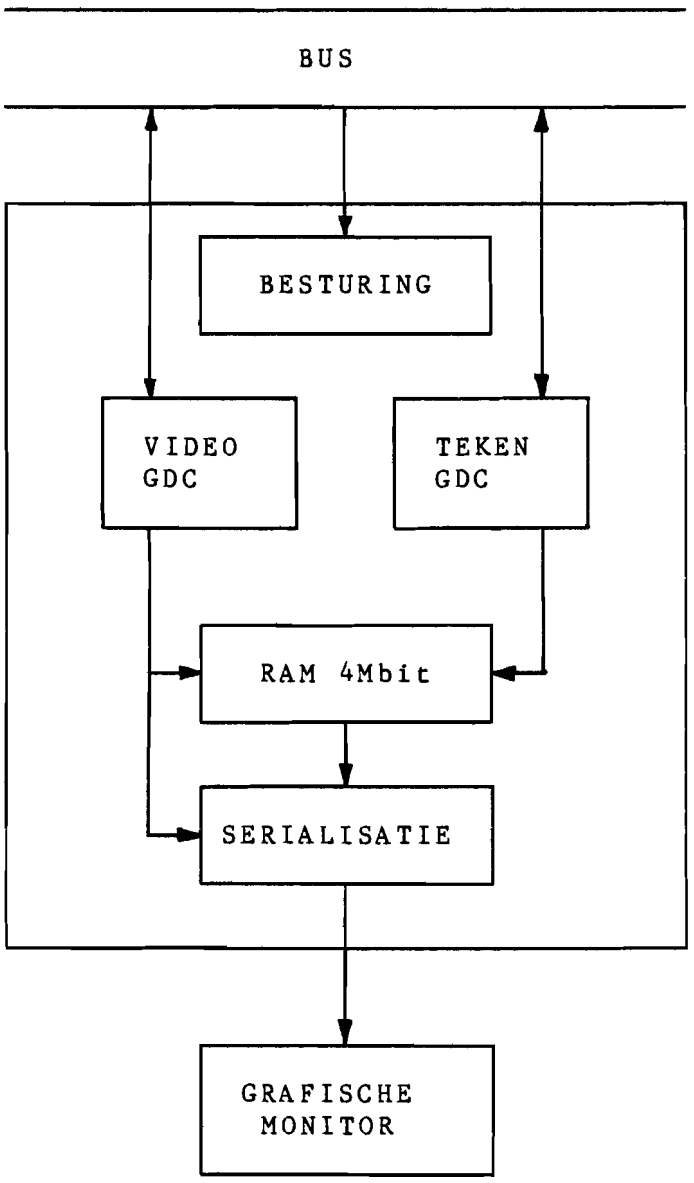

FIG 4.1 Blokschema grafische kaart

In figuur 4.1 is een blokschema getekend van de grafische kaart. De grafische kaart ontvangt zijn commando´s van de bus. Op de kaart bevinden zich twee GDC´s, een video GDC en een teken GDC. De teken GDC moet de te tekenen figuren intekenen in het RAM van 4 megabit. De video GDC heeft enkel als taak om het RAM continu uit te lezen. Zodoende vindt er een continue afbeelding plaats van de tekening die in het RAM staat, op de grafische monitor. Hiertoe wordt het RAM woord voor woord uitgelezen. Ieder woord wordt ingeklokt in

de serealisatie schakeling en door deze schakeling bit voor bit, d.w.z. pixel voor pixel, doorgestuurd naar de grafische monitor. Aangezien er met kleuren wordt gewerkt, is het RAM onderverdeeld in vier parallelle bitplanes. Ieder bitplane is verbonden met een eigen serealisatie schakeling, zodat uiteindelijk ieder pixel op de grafische monitor bestaat uit een combinatie van vier bits.
Verder is er nog een stuk besturing aanwezig op de grafische kaart. Deze besturingsregisters worden geladen via de bus. Ze kunnen echter niet uitgelezen worden. Alle informatie die vanuit de grafische kaart uitgelezen moet worden, kan alleen vanuit het statusregister van de GDC's uitgelezen worden. De besturingshardware bestaat o.a. uit een aantal registers, die onder meer de beeldvergrotingsfactor bepalen. Tevens kan hiermee de kaart in verschillende modes geschakeld worden: De character mode voor het tekenen van alpha-numerieke characters, en de normal- en linestyle mode voor het tekenen van lijnen en andere figuren.
Tenslotte zijn er nog een aantal tabellen op de kaart aanwezig, namelijk de zogenaamde modify tabel en de colour look-up tabel. De colour look-up tabel dient er voor om de logische kleuren, die in het grafische RAM staan, om te zetten in werkelijke kleuren. Hiertoe fungeert de logische kleur in het grafische RAM als een wijzer naar een plaats in de colour look-up tabel, waar dan de werkelijke kleur staat. De modify tabel speelt een centrale rol bij het tekenen van figuren in RAM en zal hieronder aan de hand van figuur 4.2 nader verklaard worden. Voor een gedetailleerde beschrijving van de grafische kaart wordt verwezen naar lit. 1) afstudeerverslag M.L. Verhoef.

4.2 Tekenen met de Grafische Kaart

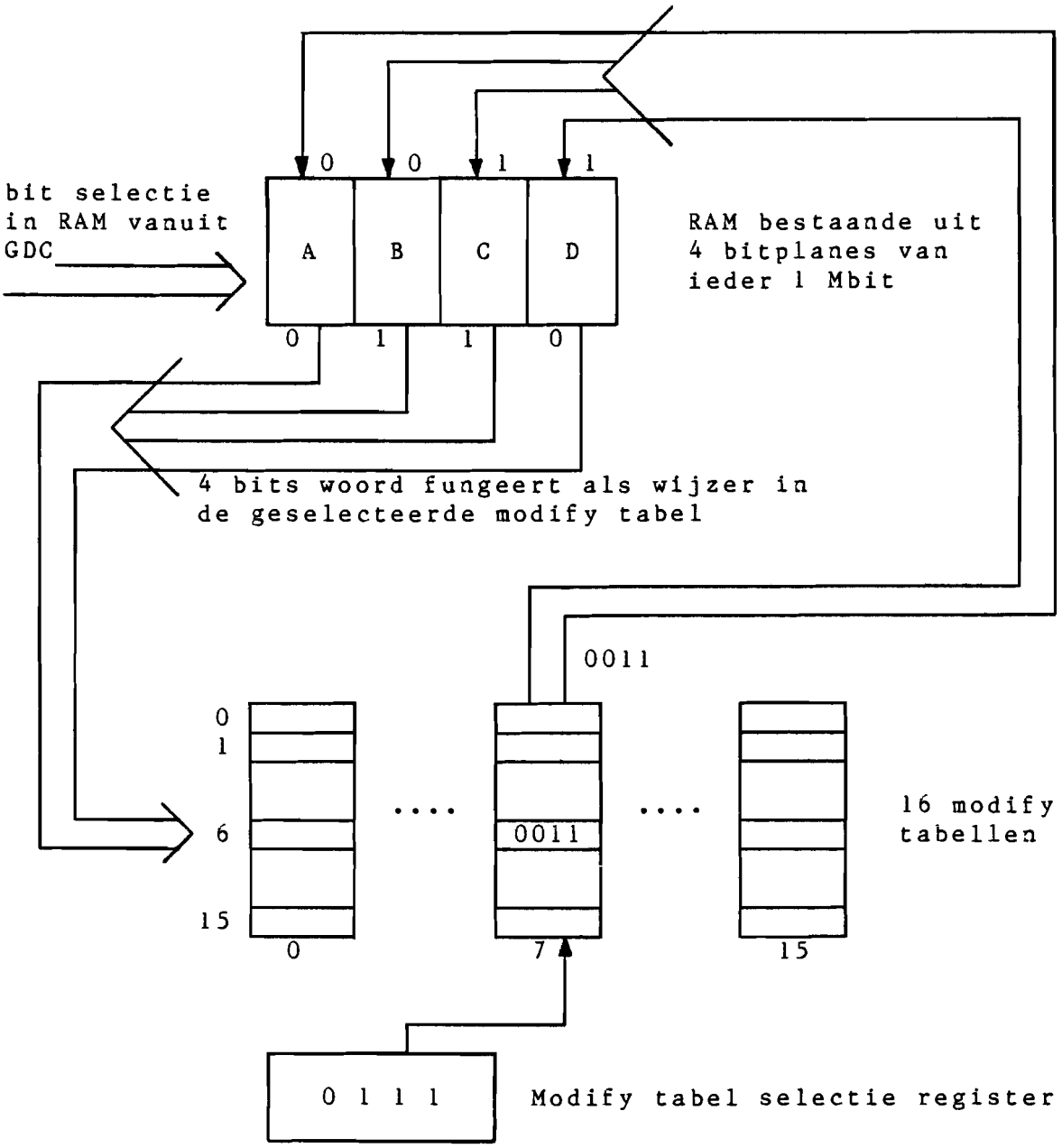

FIG 4.2 Tekenen van figuren in RAM d.m.v. terugkoppellus

De werking van de grafische kaart wijkt af van de werking van de gangbare conventionelel systemen. Indien men een grafisch systeem met kleur wil verwezenlijken, dan zal het grafische bitmap geheugen onderverdeeld moeten worden in meerdere bitplanes. De kleur van het pixel op het grafische beeldscherm wordt bepaald door de, bij dat pixel behorende, kleurindex. Indien men bijvoorbeeld werkt met een grafisch systeem bestaande uit vier bitplanes, dan zal iedere kleur-index bestaan uit vier bits, zodanig dat ieder bitplane precies een bit levert. De vier bitplanes staan parallel

geschakeld. Indien een bepaald pixel wordt uitgelezen uit het grafisch geheugen, dan wordt uit ieder van de vier parallelle bitplanes dat bit uitgelezen, dat het adres heeft dat bij het betreffende pixel op het grafische beeldscherm behoort. Op deze wijze levert ieder bitplane dus een bit voor de kleurindex van het uitgelezen pixel. Indien echter een bepaald pixel in het grafisch geheugen ingetekend moet worden, moet in de conventionele systemen achtereenvolgens in alle vier de bitplanes het betreffende bit geset of gereset worden. De vier bij een pixel behorende bits kunnen dus wel parallel uitgelezen worden, maar moeten achtereenvolgens per bitplane worden ingesteld.

In het op de vakgroep EB ontwikkeld grafisch systeem, worden bij het schrijven in het geheugen alle vier de bitplanes tegelijkertijd geadresseerd. Een bepaald pixel kan zodoende in een keer op een betreffende kleur worden ingesteld. In figuur 4.2 is aangegeven hoe dit in grote lijnen gebeurt.
De functie van de teken GDC (zie figuur 4.1) die in conventionele ontwerpen de bits in een bitplane uitleest en intekent, d.w.z. reset of set, is hier gewijzigd. De GDC kan in dit ontwerp geen bits meer setten en resetten, maar zorgt er enkel en alleen voor, dat het betreffende bit geadresseerd wordt. Als er dus een lijn in het geheugen getekend moet worden, doet de GDC niets anders dan achtereenvolgens de juiste bits adresseren. Het tekenen gebeurt dan automatisch door middel van een terugkoppellus, waarin een van de zestien modify tabellen is opgenomen. Het tekenen kan op twee manieren plaats vinden: In de "multipixel" mode en in de niet-"multipixel" mode.

Tekenen in de Niet-multipixel Mode

Als een bepaald pixel ingetekend moet worden, dan zal de GDC er voor zorgen dat de betreffende bits in de vier parallelle bitplanes geadresseerd worden. Aan de uitgang van de vier bitplanes zullen nu de respectievelijke bits verschijnen. In het voorbeeld op figuur 4.2 wordt uit bitplane A een 0 uitgelezen, uit bitplane B een 1, uit bitplane C een 1 en uit bitplane D een 0. Deze vier bits vormen tezamen de kleurindex van het betreffende pixel. De kleurindex is in dit geval dus 6. Wat wordt nu de nieuwe kleurindex van dit pixel? Welnu, het vier bits woord (de kleurindex) gaat fungeren als een wijzer, die in de geselecteerde modify tabel een plaats aanwijst. De totale modify tabel is een RAM dat is opgebouwd uit zestien parallelle modify (sub)tabellen. Iedere tabel bevat zestien plaatsen met indexen 0 t/m 15. Iedere plaats bevat een woord van vier bits. Alle plaatsen van iedere modify tabel (in totaal dus 16 x 16 = 256 plaatsen) kunnen vanaf de bus (zie figuur 4.1) geladen worden. Bij de modify tabellen behoort een modify tabel selectieregister, dat eveneens via de bus geladen kan worden. Dit register selecteert, of wel activeert, een bepaalde tabel van de zestien tabellen. In dit geval is tabel nummer 7 geselecteerd. De kleurindex van het uitgelezen pixel zal in de aktieve tabel, in dit geval tabel nr 7, een plaats selecteren. In dit geval dus plaats nr 6. Het vier bits

woord dat op deze geselecteerde plaats staat, zal de nieuwe kleurindex van het geselecteerde pixel worden. Hiertoe wordt een verbinding tot stand gebracht van de geselecteerde plaats in de geselecteerde modify tabel, naar het grafische 4Mbit RAM. Via deze verbinding zullen de vier bits uit de parallelle bitplanes, die door de GDC geadresseerd worden, worden ingelezen. De kleurindex van het geselecteerde pixel zal in dit geval dus wijzigen van 6 naar 3.
Het aantal plaatsen dat iedere modify tabel bevat, moet uiteraard gelijk zijn aan het aantal verschillende kleurindexen, dat met de vier bitplanes gevormd kan worden. Door nu de zestien plaatsen van de geselecteerde modify tabel op een bepaalde manier te laden, kunnen verschillende tekenfuncties gerealiseerd worden. In alle gevallen zal de nieuwe kleur van een ingetekend pixel, afhankelijk zijn van de oude kleur, die het pixel voor het tekenen bezat.

Tekenen in de Multipixel Mode

Wanneer we echter gaan tekenen in de multipixel mode, dan zal de terugkoppellus onderbroken worden. De nieuwe kleur van een pixel zal nu niet meer afhankelijk zijn van zijn oude waarde. Het woord multipixel moet in die zin begrepen worden, dat alle nu ingetekende pixels dezelfde kleur krijgen. In de multipixel mode wordt de kleurindex van het geadresseerde pixel niet meer gebruikt als wijzer, die een plaats in de geselecteerde modify tabel selecteert. In plaats hiervan wordt altijd plaats nr 15 (hexadecimaal 0x0f) geselecteerd. D.w.z. dat de nieuwe kleur van het ingetekende pixel de kleur wordt, die is opgeslagen in de geselecteerde modify tabel op plaats nr 15 (0x0f). Het tekenen in de multipixel mode is dus analoog aan het tekenen in de niet-multipixel mode waarbij alle 16 plaatsen van de geselecteerde tabel dezelfde waarde (kleurindex) hebben.
Voor normale toepassingen wordt getekend in de multipixel mode. De niet-multipixel mode wordt onder meer gebruikt, indien men in een bitplane wil tekenen, zonder de grafische informatie in de overige bitplanes te wijzigen.

4.3 M68000 Systeem

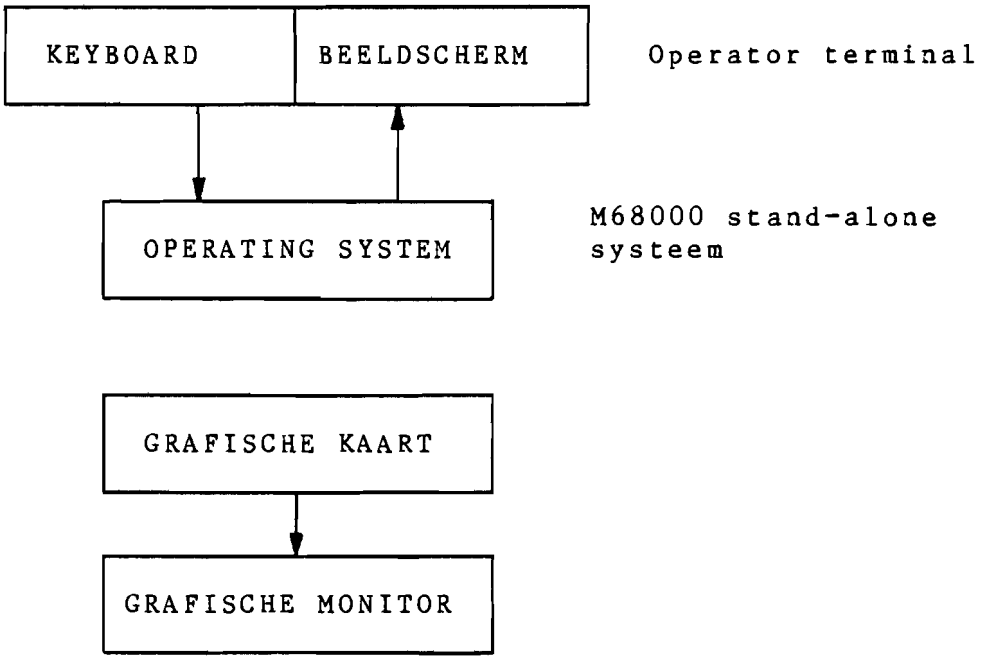

FIG 4.3 M68000 systeem met grafische kaart

Het grafische systeem is momenteel alleen nog maar operationeel in het M68000 systeem. Dit systeem is weergegeven in figuur 4.3. Het is een stand-alone systeem, dat bestaat uit een M68000 processor kaart, een geheugenkaart en de grafische kaart. Het operating systeem is een simpel monitor programma met enkele letter commando's. Het operating systeem haalt zijn input van het keyboard en pleegt output naar het beeldscherm. We kunnen nu aan het operating systeem opdracht geven, het grafische programma vanuit een diskette naar het geheugen te laden. We geven dan de controle over aan de grafische software module. We komen dan in figuur 4.4.

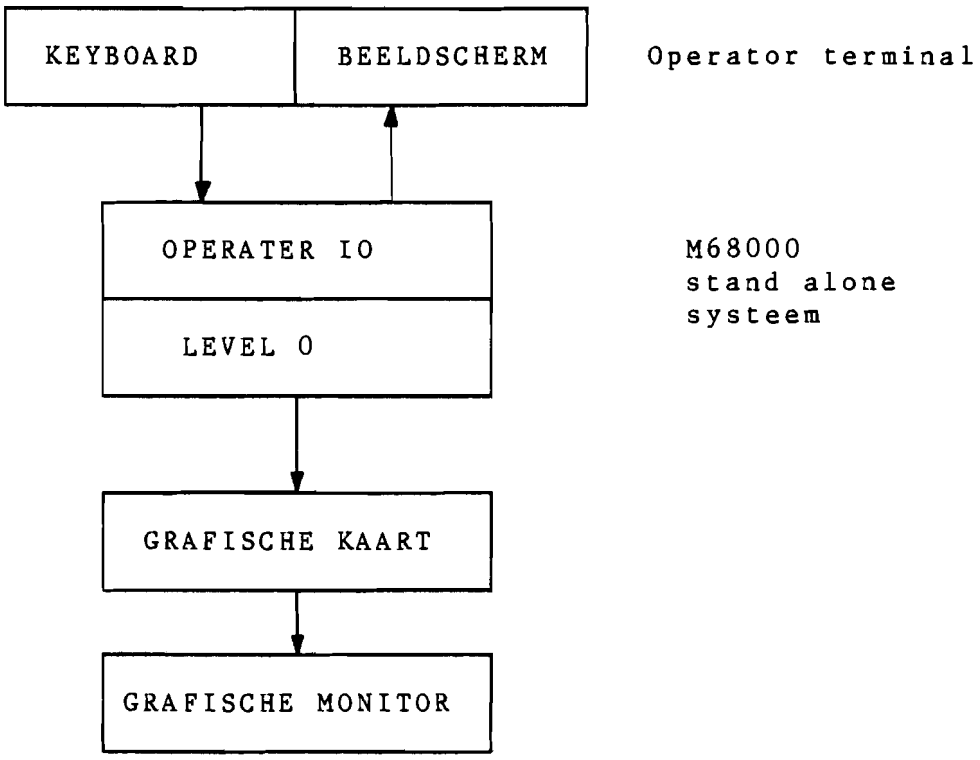

FIG 4.4 M68000 systeem als grafisch systeem

De grafische software module bestaat uit een blok OPERATOR IO en een blok LEVEL 0. De operator die achter het keyboard zit, typt de commando's in. Deze commando's worden door de OPERATOR IO software naar binnen gehaald. Er wordt gekeken of het een bestaand commando is. Zo ja, dan wordt de output op het beeldscherm gebracht. Zo nee, dan wordt er een error message op het beeldscherm gebracht. De commando's zijn in het algemeen een letter commando's, die eventueel gevolgd kunnen worden door heximale of decimale input. De gedecodeerde commando's worden door het blok OPERATOR IO doorgegeven naar de LEVEL 0 software. De LEVEL 0 software zet de commando's om in elementaire aansturing van de grafische kaart. De grafische kaart vervolgens, pleegt output naar de grafische monitor. De hier besproken software is in het M68000 systeem geheel uitgetest en operationeel.

4.4 Unix

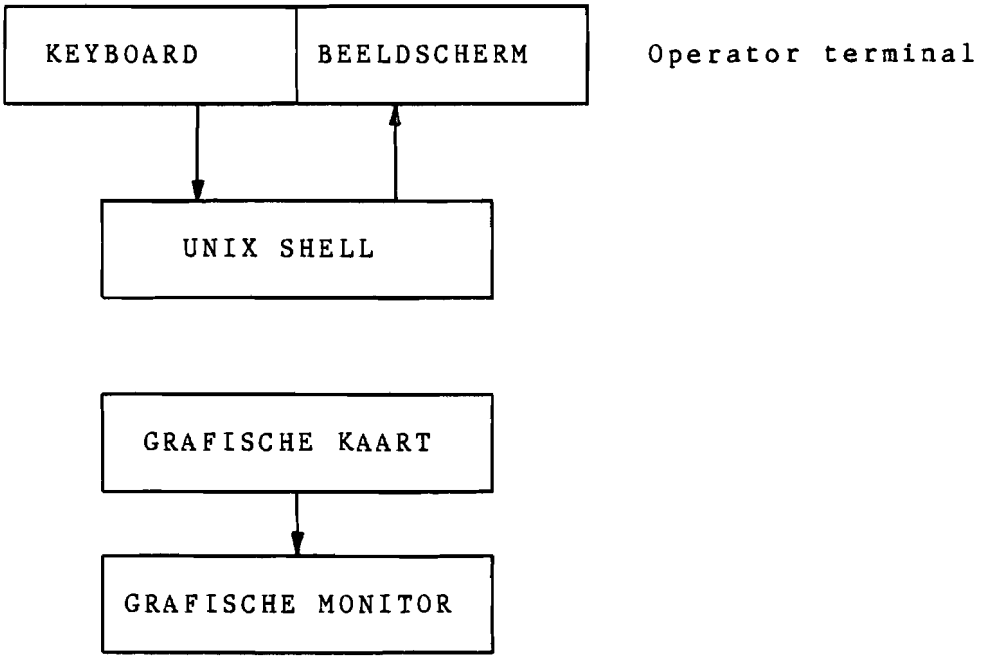

FIG 4.5 Unix systeem met grafische kaart

Het uiteindelijke doel van het grafische systeem is werking onder Unix. In figuur 4.5 is een simpele opbouw van het grafische systeem onder Unix getekend. We zien dat dit systeem vrijwel identiek is aan het M68000 systeem. Alleen het operating systeem is nu de Unix shell in plaats van de M68000 monitor. De Unix shell kan echter nog niet de grafische kaart aansturen. Hiertoe moet eerst de grafische software geladen worden, zodat we in figuur 4.6 terecht komen. Figuur 4.6 is vrijwel identiek aan figuur 4.4. Het verschil hierbij is echter, dat in het M68000 systeem de output naar de grafische kaart op zeer eenvoudige wijze plaats kan vinden. Namelijk gewoon via memory mapped IO. Onder Unix is dit niet zonder meer mogelijk, aangezien alle IO op file basis gebeurt. Men kan dus alleen in- en output plegen naar en van files. Voor speciale toepassingen kan echter gebruik gemaakt worden van de /dev/mem file. Wanneer men naar deze file output pleegt, doet men in feite niets anders dan direct output plegen naar het Unix systeem geheugen. De inhoud van de /dev/mem file is gemapped op het Unix systeem geheugen. We kunnen via een lseek operatie de pointer binnen de file op een bepaalde plaats instellen en hierna output plegen naar de file. De output naar deze file wordt dan doorgeleid naar de geselecteerde geheugenplaats. Op analoge wijze kan er ook input plaats vinden vanuit het geheugen.
We moeten de grafische kaart dus ergens in het Unix systeem geheugen plaatsen. De adressen van de verschillende registers en van de GDC's moeten bekend zijn. We kunnen dan via

lseek operaties de grafische kaart adresseren en dan in- en output doen.

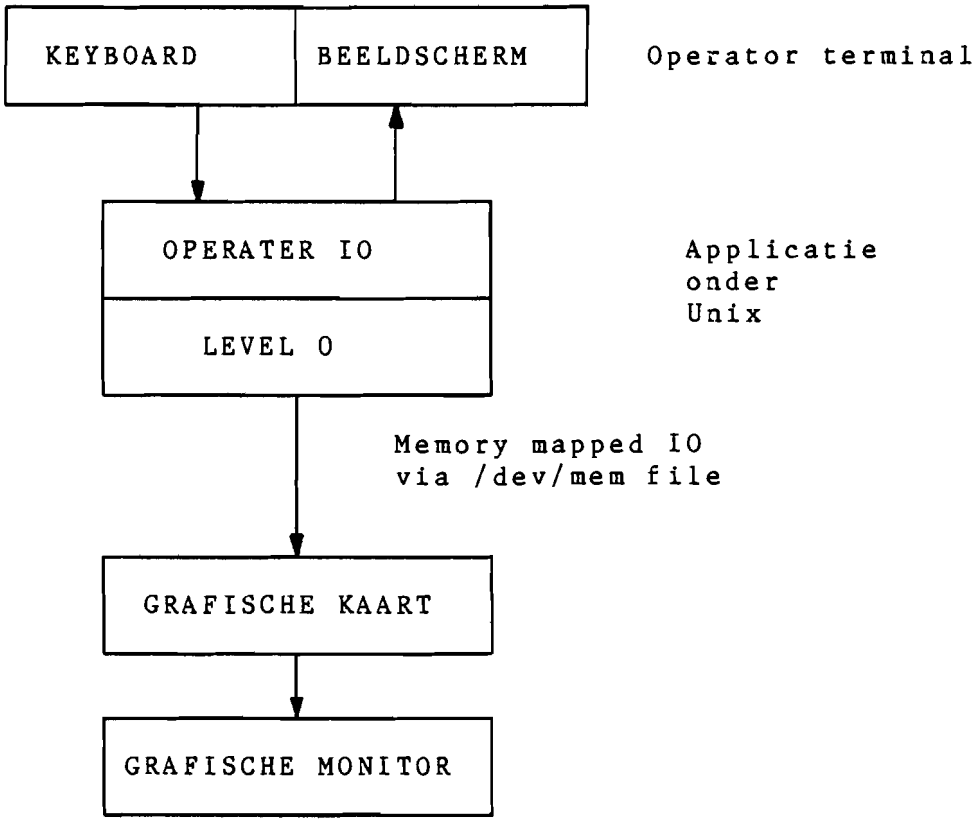

FIG 4.6 Grafisch testsysteem onder Unix

Het uiteindelijke doel is de implementatie van GKS op het grafische systeem. In figuur 4.7 is de GKS applicatie onder Unix in zijn eenvoudigste vorm getekend. De software laag LEVEL 0 is identiek aan die van figuur 4.6. De driver is een laag software die de interface verzorgt tussen GKS en de LEVEL 0 software. De GKS laag is een verzameling van grafische procedures, die door de GKS applicatie kunnen worden aangeroepen, en die output plegen naar de driver. De GKS applicatie is een programma dat bijvoorbeeld lijnen kan tekenen, tekst kan tekenen, figuren kan tekenen etc.. De GKS applicatie haalt input vanuit het keyboard. De input wordt geechood op het beeldscherm. De grafische figuren komen uiteindelijk op de grafische monitor terecht. Voor een nauwkeurigere omschrijving van GKS wordt verwezen naar hoofdstuk 7.
Tijdens mijn afstuderen is de software uit figuur 4.6, en uit figuur 4.7 de LEVEL 0 software en de DRIVER laag, door mij ontwikkeld. Deze software is echter nog niet uitgetest onder Unix, aangezien het Unix systeem nog niet operationeel was. De GKS software is een GKS implementatie geschreven in C door het Centrum voor Wiskunde en Informatica te Amster-

dam. Bij dit pakket behoren ook enkele testprogramma's, die als GKS applicatie gebruikt kunnen worden. Het is duidelijk dat voor de verschillende applicaties uit figuur 4.4, 4.6 en 4.7 verschillende software modules nodig zijn. De software voor deze modules staat zo veel mogelijk in dezelfde files. Door middel van conditionele compilatie kunnen de verschillende modules geheel geautomatiseerd worden aangemaakt.

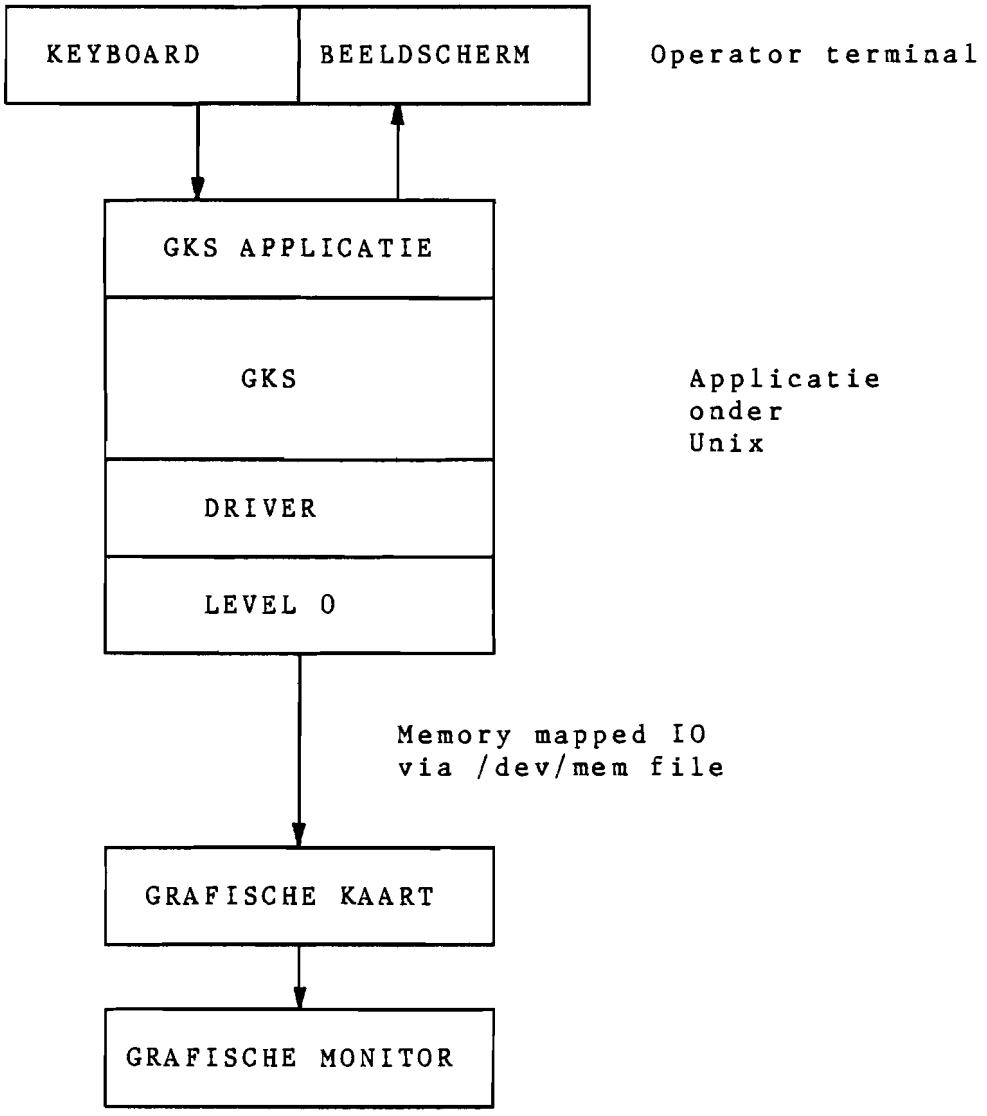

FIG 4.7 Eenvoudige GKS implementatie onder Unix

4.5 Toekomstig Systeem

Het ligt voor de hand dat de oplossing van figuur 4.7 nog niet de definitieve oplossing is. Een nadeel van deze oplossing is, dat de LEVEL 0 software als applicatie onder Unix draait. Hierdoor wordt Unix door een GKS applicatie zeer zwaar belast. De interactie tussen de LEVEL 0 software en de

grafische kaart is namelijk zeer intensief. Om bijvoorbeeld een lijn te tekenen op de grafische monitor, moeten er elf bytes achtereenvolgens naar de teken GDC gestuurd worden. Voordat iedere byte verstuurd kan worden, moet eerst worden getest of de GDC klaar is om de volgende byte te ontvangen. Tevens moet, indien men bepaalde registers op de grafische kaart wil laden, gewacht worden totdat de GDC klaar is met het tekenen, aangezien anders het tekenproces verstoord zou worden. Het gevolg hiervan is dat de LEVEL 0 software kontinu moet testen of de GDC klaar is, hetgeen een enorme overhead veroorzaakt.
Een tweede nadeel is, dat de operator aan twee beeldschermen tegelijkertijd moet werken. Op het ene beeldscherm worden zijn ingevoerde commando's geechood en op het tweede beeldscherm, de grafische monitor, worden de grafische figuren zichtbaar. Als het systeem volgens figuur 4.7 geheel is uitgetest, zal het systeem dan ook verder geperfectioneerd moeten worden. Dit nieuwe systeem wordt hierna behandeld.

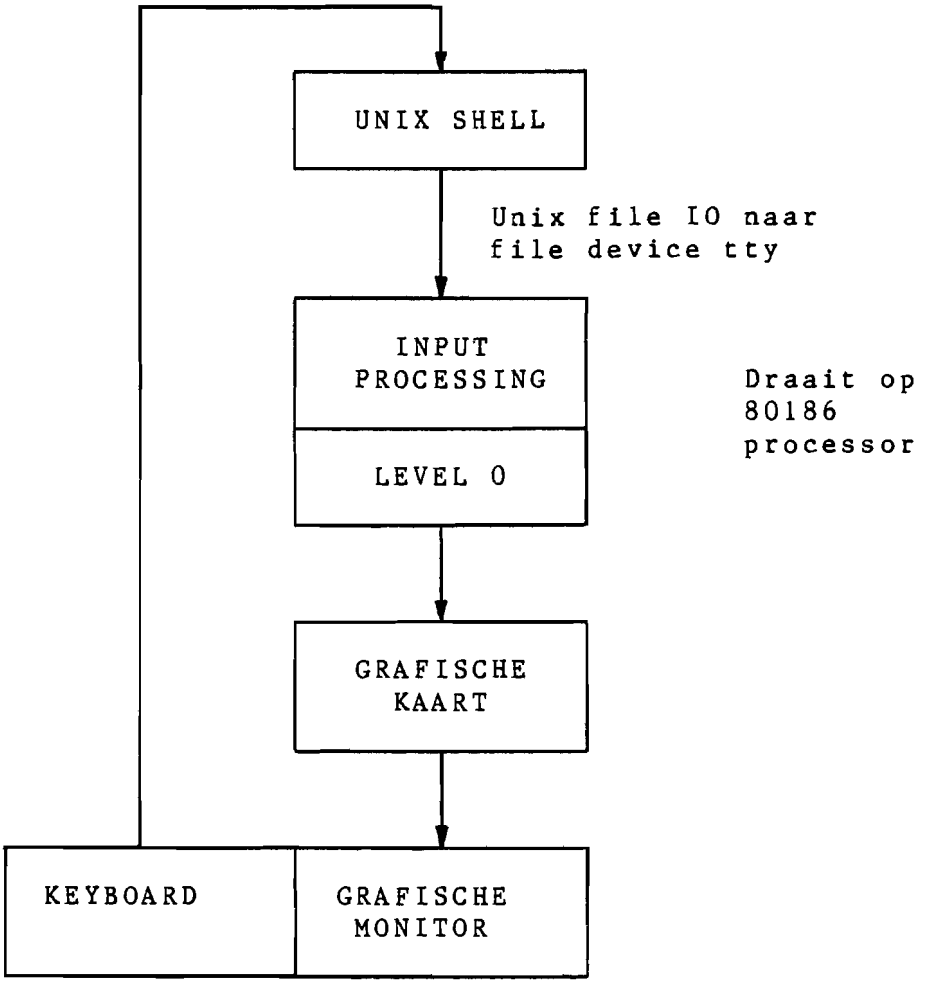

FIG 4.8 Grafisch werkstation als Unix terminal

In figuur 4.8 is het in de toekomst te ontwikkelen grafische systeem geschetst. We zien hier dat er geen twee beeldschermen meer aanwezig zijn. De grafische monitor is nu aan het keyboard gekoppeld. Voor deze oplossing is een extra processorkaart nodig. Dit zou bijvoorbeeld een 80186 processor kunnen zijn waar de software blokken INPUT PROCESSING en LEVEL 0 op draaien. De 80186 kaart moet via een interne bus verbonden zijn met de grafische kaart. De LEVEL 0 software is gelijk aan de eerder genoemde LEVEL 0 modules en stuurt de grafische kaart aan. De grafische kaart stuurt op zijn beurt de grafische monitor aan. Het totale blok bestaande uit de 80186 kaart, de grafische kaart, de grafische monitor en keyboard, is een grafisch werkstation en moet zich gedragen als een tty. Tty stamt nog uit vroeger tijden en is een afkorting voor teletype. Hoewel nu beeldscherm terminals de functie van teletypes hebben overgenomen, is de afkorting tty nog blijven bestaan. Dit grafische werkstation is dus in feite niets anders dan een normale Unix terminal, die tevens grafische mogelijkheden heeft. De software laag INPUT PROCESSING moet er voor zorgen dat het grafische werkstation zich naar buiten toe gedraagt als een Unix tty. D.w.z. dat Unix output naar het grafische werkstation kan doen op basis van file IO naar een tty file device. Als een Unix gebruiker op de grafische terminal inlogt onder Unix, moet de input processing zodanig ingesteld zijn, dat de hele grafische software transparant wordt. De input, d.w.z. de Unix commando's die op het keyboard worden ingetypt en door de Unix shell worden geechood, moet dan op het grafische scherm terecht komen. Als het commando wordt gegeven om een GKS applicatie uit te voeren, laadt Unix de grafische software en komen we in figuur 4.9 terecht.

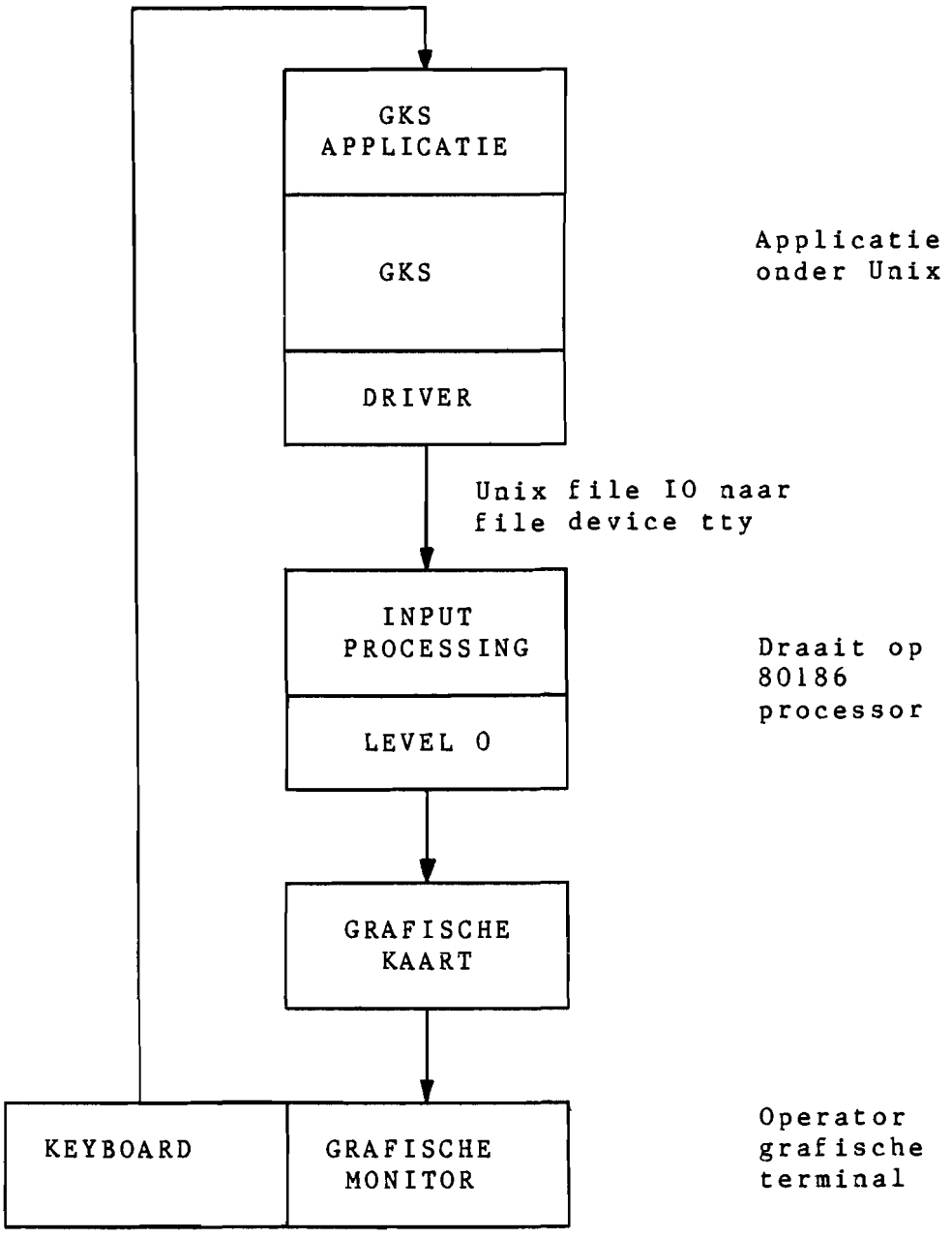

FIG 4.9 GKS applicatie onder Unix met grafisch werkstation

We zien hier een GKS applicatie die gelinkt is met de GKS software en eindigt op een driver. Een van de eerste dingen die de GKS applicatie moet doen, is het openen van het aangesloten werkstation. De driver moet daartoe de juiste commando´s doorgeven aan de INPUT PROCESSING software op de 80186 kaart, zodat het grafische werkstation omgesteld wordt naar de grafische mode.
Aangezien de LEVEL 0 software nu op een aparte processor draait, is een groot deel van de belasting van Unix weggevallen. Tevens kunnen op deze wijze op een zeer eenvoudige

manier meerdere grafische werkstations worden aangesloten. Als we op het Unix systeem een tweede grafisch werkstation aansluiten, behoeft de GKS applicatie enkel en alleen te weten, onder welke files dit grafisch werkstation te bereiken is. De GKS applicatie kan dan het tweede werkstation eveneens openen. Hierbij moet aan GKS doorgegeven worden, wat voor type werkstation het is en als welk tty file device het werkstation bereikbaar is. Indien het tweede werkstation een ander type is dan het eerste, dan zal er naast de eerste driver nog een tweede driver meegelinkt moeten worden. De driver(s) kan (kunnen) nu naar beide werkstations output plegen, eenvoudigweg door naar twee verschillende files output te plegen.

5 Kwaliteitseisen voor de Software Ontwikkeling

Om een goede kwaliteit van de ontwikkelde software te waarborgen, is er bij de software ontwikkeling speciaal aandacht besteed aan de volgende punten:

- In de praktijk blijkt dat het commentaar in de software listings meestal een sluitpost is. Zo wordt er bijvoorbeeld in de GKS listings slechts sporadisch gebruik gemaakt van commentaar. Toch is het commentaar een essentieel onderdeel van software sources. Software sources moeten een op zichzelf staand geheel vormen. Bij de bestudering van de software listings, moet het commentaar in de listings zodanig zijn aangebracht, dat de listings in principe zonder verdere documentatie bestudeerd kunnen worden.
  Het programma is dan ook ruim voorzien van commentaar. In het verslag is alleen de globale opzet van de software besproken.

- De source files zijn zodanig geedit dat er een overzichtelijke en consekwente lay-out ontstaat. De sources zijn hierdoor makkelijk leesbaar.
  Ingewikkelde constructies die mogelijk zijn in de programmeertaal C en alleen door de schrijver van de software begrepen kunnen worden, zijn vermeden. Het programma is logisch en structureel van opbouw. In dit verband is geprofiteerd van de bestudering van de GKS software. Na de bestudering van de GKS software, zijn in de zelf ontwikkelde software nog verschillende structurele wijzigingen aangebracht.

- Er is naar gestreefd de software zodanig te schrijven, dat er een zo groot mogelijke mate van portability (overdraagbaarheid) wordt verkregen. D.w.z. dat de specifieke hardware afhankelijkheid van het programma zoveel mogelijk is beperkt. Hiertoe is bijvoorbeeld alle IO naar de grafische kaart teruggebracht tot slechts drie functies: Getbyte, putbyte en putword. Door middel van conditionele compilatie kunnen deze drie functies worden aangepast aan het systeem waarop de software gedraaid wordt. Zo is ook voor toekomstige ontwikkelingen het programma makkelijk aanpasbaar voor andere hardware. Constructies die eventueel problemen zouden kunnen opleveren in betrekking tot de portability, zijn zoveel mogelijk vermeden. Indien toch toegepast, is er door middel van commentaar extra op geattendeerd.

- Ten slotte is er bij de ontwikkeling van de software ook gezorgd voor een goede ontwikkelomgeving. D.w.z. hulpmiddelen om de software ontwikkeling zoveel mogelijk te automatiseren. Dit bevordert de kwaliteit van de ontwikkelde software. Tevens biedt het een belangrijke ondersteuning voor toekomstige afstudeerders, die de software verder zullen moeten uitbouwen. De toegepaste ontwikkelomgeving is in detail behandeld in hoofdstuk 9.

6 Software voor de Grafische Kaart

6.1 Lokalisatie van de Files

De software voor de grafische kaart bevindt zich op het Unix systeem van de TH Eindhoven (Unix VAX in Rekencentrum). De software staat onder inlognaam elebst3. De directory van deze inlognaam is als volgt georganiseerd:
In de directory bin staan alle zelfgeschreven hulputilities, waaronder: shellprocedures, C programma's, awk programma's en sed programma's. In de directory graf staat alle software voor de grafische kaart en voor de GKS driver. In de directory gks staan alle GKS files van de C Implementatie van het Centrum voor Wiskunde en Informatica. In de directory etc ten slotte staan vier files die gebruikt worden bij het uitprinten van de GKS files uit de directory gks.

6.2 Listings

In bijlage A zijn de software listings bijgevoegd. Deze listings zijn als volgt opgebouwd. Op het eerste blad, het titelblad staat de datum en tijd van uitdraai. De data die op de volgende bladen vermeld staan, zijn de data waarop de files, die op dat moment geprint worden, gecreeerd zijn, of voor het laatst gewijzigd zijn. Als eerste is de makefile uit de directory graf geprint. Vervolgens zijn de files geprint die benodigd zijn voor de software voor de OPERATOR IO en de LEVEL 0 modules (zie hoofdstuk 4). Dit zijn de files die in de makefile gedefinieerd zijn als SOURCES. Als eerste zijn de header files gelist. De header files zijn files met achtervoegsel .h. Deze files worden geinclude in de c files, de files met achtervoegsel .c. De header files zijn op alfabetische volgorde gelist. De c files zijn onderverdeeld in drie niveau's. De niveau's 0, 1 en 2. Het niveau van een source file wordt aangegeven door het getal dat direct voor het achtervoegsel .c staat. Als regel is aangehouden dat de functies die gedefinieerd zijn in een file van een bepaald niveau, alleen maar functies mogen aanroepen van hetzelfde niveau of lager. Bijv. een functie die gedefinieerd is op niveau 1 mag alleen functies aanroepen uit niveau 1 en uit niveau 0. De c files van niveau 2 bevatten alleen functies die de in- en output met het keyboard en de alpha-numerieke terminal verzorgen, d.w.z. de interactie met de operator (niet met de grafische terminal). Dit niveau wordt in de figuren uit hoofdstuk 4 aangeduid als de OPERATOR IO laag software. Na de header files worden de files van niveau 2 gelist. De listing begint met de file main2.c, aangezien dit het beginpunt van de software is, en vervolgt met de files van niveau 2 in alfabetische volgorde. Hierna worden achtereenvolgens de files van niveau 1 in alfabetische volgorde en de files van niveau 0 in alfabetische volgorde gelist. De software van niveau 1 en 0 wordt in hoofdstuk 4 aangeduid als de LEVEL 0 laag.
Vervolgens worden de files voor de GKS driver gelist. Deze files staan eveneens in de directory graf en beginnen allen

met de letters geb, afkomstig van Grafisch systeem van de vakgroep EB. Van deze files worden allereerst de header files weer gelist op alfabetische volgorde en vervolgens de enige c file.
Aaneensluitend aan de listing van de files, worden alle in deze files gedefinieerde procedures gelist, met daarachter de file naam waarin zij gedefinieerd zijn en het regel nummer in de file.
Analoog als met de gedefinieerde procedures zijn ook alle #define statements gelist. De define statements staan op alfabetische volgorde van de gedefinieerde variabelen, en worden vooraf gegaan door de file naam waarin zij gedefinieerd zijn met bijbehorend regelnummer uit de file.
Vervolgens worden van alle subdirectories de namen van de aanwezige files gelist, behalve uit de directory gks aangezien hier alle files staan van de GKS implementatie van het C.W.I.. De listing wordt afgesloten met een listing van alle files die gedefinieerd zijn in de subdirectory bin. Aangezien de meeste van deze files slechts uit enkele regels bestaan, zijn ze op een gewijzigde manier gelist. Iedere file wordt voorafgegaan door een opeenvolging van = tekens met daartussen de file naam. Verder zijn de regels genummerd en worden alle files, gescheiden door twee nieuwe regels, achtereenvolgens geprint.

6.3 Globale Opzet van de Software

De software is op de volgende manier opgezet. Alle c files beginnen met de statement #include "include.h". De include.h file is een file die zelf alleen maar bestaat uit include statements van een zevental files. Op deze manier wordt er voor gezorgd, dat in elke c file de zeven header files worden geinclude. Een van de zeven header files is de file options.h. In deze file staan alle opties die de conditionele compilatie bepalen. Als een bepaalde module gecompileerd moet worden, worden de c files gecompileerd, gelinkt en geladen tot een module. In de c files staan echter verschillende conditionele compilatie statements. Het gehele mechanisme van de conditionele compilatie is verklaard in de file options.h en in hoofdstuk 9.2 Makefile.
De software is zo hardware onafhankelijk mogelijk opgezet. Alle in- en output functies naar de grafische kaart zijn geschreven in termen van macro's. De macro's zijn gedefinieerd in de file macros.h. Alle in- en output macro's zelf zijn weer herleid tot drie basis functies, namelijk getbyte putbyte en putword. Deze drie basis functies zijn door middel van conditionele compilatie aangepast aan het systeem. Indien er voor het M68000 systeem gecompileerd moet worden, dan worden deze functies als memory mapped IO gedefinieerd. Indien er voor het Unix systeem gecompileerd moet worden, dan worden deze functies gedefinieerd als Unix system calls. Op deze manier is de gehele software ook zeer eenvoudig aan te passen aan andere systemen, zoals bijvoorbeeld de 80186 processor. Hiertoe volstaat het om alleen de drie basis functies getbyte putbyte en putword te herdefi-

nieren op basis van 80186 functies.
In de taal C is het mogelijk om output te doen naar bepaalde geheugen plaatsen, zonder dat hiervoor assemblers routines hoeven te worden geschreven. De definitie van de output functies getbyte putbyte en putword heeft van dit mechanisme gebruik gemaakt.
In de file hardaddr.h is een soortgelijk mechanisme gebruikt om de in- en output functies van het M68000 monitor programma, namelijk printf, getchar en putchar, te gebruiken. Deze functies staan op vaste adressen in het monitor ROM programma. Zij kunnen door middel van bepaalde definitie statements in C rechtstreeks worden aangeroepen, zonder dat hierbij assemblers routines nodig zijn. Deze definities staan in de file hardaddr.h.
De taal C biedt de mogelijkheid tot het opstellen van eigen type definities. Hiervan is gebruik gemaakt in de file typedefs.h.
Het is algemeen bekend dat bij het programmeren een overvloedig gebruik van globale variabelen dient te worden vermeden. Er zijn in dit programma dan ook slechts enkele globale variabelen gebruikt, welke gedeclareerd zijn in de file globals.c. Bovendien zijn de meeste van deze globale variabelen bijeen gebracht in een structure. Deze structure isl (intermal state list) bevat alle voor het programma belangrijke parameters. Een voordeel van het onderbrengen van de gebruikte parameters in een centrale structure, is naast de overzichtelijkheid ook het volgende: Wanneer ergens in het programma gerefereerd wordt aan een variabele die member is van de structuur isl, dan is het direct duidelijk, dat dit een globale variabele is. De type definitie van de structuur isl staat in de file typedefs.h. Ook de parameters van de aangesloten grafische monitor zijn in de internal statelist opgenomen, als de structure term van het type videoterm.

6.4 Commando Interpreter

Bij applicaties zonder GKS, waarbij dus alle files van niveau 2 worden gecompileerd en meegelinkt, begint het programma met de main module uit de file main2.c. De main module accepteert eenletterige input. Deze input wordt geechood naar de terminal. Als het foutieve input is, wordt er geechood "illegal command". Indien de input correct is, wordt het, bij die input letter behorende, commando uitgevoerd. Het kan zijn dat er na de eerste input letter nog meerdere input volgen moet, zoals bijvoorbeeld een hexadecimaal getal of decimale getallen. Voor de verwerking van de volgende input worden er altijd subroutines aangeroepen. Als meerletterige commando's worden ingevoerd, mogen er tussen de letters geen spaties worden ingevoerd. Na elke letter wordt er echter wel een spatie op het scherm geechood.
Indien de input geheel compleet is, worden de grafische commando's uitgevoerd en wordt er weer teruggekeerd naar de main routine. De subroutines die eventuele vervolg input moeten verwerken, zijn altijd gedefinieerd in files op

niveau 2. De uitvoering van de grafische commando's wordt gedaan door procedures die gedefinieerd zijn op niveau 1 of niveau 0.
Alle commando's kunnen worden voorafgegaan door een willekeurig aantal spaties en nieuwe regel tekens. Hierdoor is het onder Unix mogelijk, dat de commando's worden ingevoerd in een file. Het grafische programma kan de input dan halen uit die file, in plaats van vanaf het keyboard. Op deze manier kunnen de commando's voor het tekenen van bepaalde figuren van tevoren in een file worden opgeslagen, waarna ze in een keer kunnen worden uitgevoerd. Na het opstarten van het systeem verdient het aanbeveling om het xl commando uit te voeren. Hierdoor worden de verschillende modify tabellen met vooraf bepaalde kleuren geladen en worden andere zaken geinitialiseerd.
Het commando verwerkings programma biedt de volgende mogelijkheden:

- Het vullen van de modify tabellen.
- Het afbeelden van characters op het grafische scherm, die op het keyboard worden ingetikt. De characters kunnen schuin (slanted) of recht worden afgebeeld. De relatieve grootte van de te tekenen characters kan worden ingesteld op 1 t/m 16 (Default 1).
- Het selecteren van een modify tabel die, bij het tekenen, in de terugkoppellus gebruikt gaat worden.
- Het positioneren van de cursor.
- Het instellen van de parameters van de grafische monitor vanaf het keyboard.
- Het opvragen van de huidige curser coordinaten. Deze uitvoer wordt op het beeldscherm zichtbaar.
- Het vullen van het gehele grafische geheugen met een bepaalde kleurindex.
- Het tekenen van een pixel op een bepaald coordinatenpunt.
- Het aanschakelen van de crosshair. De crosshair kan op een absoluut coordinaten punt geplaatst worden, maar kan tevens door middel van vier toetsen in elke willekeurige richting over het beeldscherm worden bewogen. De crosshair is een kruis dat over de bestaande tekening op het grafische beeldscherm wordt getekend, waarbij echter de onderliggende tekening niet gewijzigd wordt.
- Het tekenen van een cirkel met willekeurig middelpunt en straal.
- Het tekenen van een willekeurige lijn.
- Het vullen van de colour look-up tabel.
- Het uitvoeren van pixel panning. Pixel panning is het verschuiven van het window, met een pixel in horizontale richting. Het window is datgene wat er op het grafische beeldscherm zichtbaar is. Aangezien het grafische geheugen groter is dan het beeldscherm, kan niet alles dat in het grafische geheugen staat, worden afgebeeld op het beeldscherm. Om toch alles te zien wat in het grafische geheugen aanwezig is, moet het beeldscherm dus als het ware over het grafische geheugen heen schuiven. Zodoende zal er steeds een ander stuk uit het grafische geheugen zichtbaar worden. Het verschuiven van het beeldscherm in horizontale richting over het grafische geheugen heet panning. Het

verschuiven in verticale richting heet scrolling. De normale panning van het window gaat met 16 pixels tegelijkertijd. Door een speciale schakeling op de print kan er echter ook met een pixel tegelijkertijd gepanned worden.
- Het tekenen van een willekeurige rechthoek.
- Het opvullen van een willekeurig rechthoekig vlak.
- Het instellen van het window. Het window kan op een willekeurige plaats op het grafische geheugen worden ingesteld, met een nauwkeurigheid van 1 pixel. Hiertoe kunnen de coordinaten van de linkerbovenhoek van het window worden opgegeven. Het is echter ook mogelijk door middel van vier toetsen het window steeds met een stapje in horizontale of verticale richting (pannen of scrollen) over het grafische geheugen te bewegen.
- Het instellen van een tweede window. De GDC's maken het mogelijk om twee onafhankelijke windows tegelijkertijd op het scherm te projecteren. Deze windows liggen onder elkaar en verdelen het beeldscherm in twee horizontale gedeelten. Het aantal beeldlijnen van beide windows kan worden ingesteld, alsmede de plaats waar de windows zich in het geheugen bevinden. Het tweede window werkt echter alleen in de noninterlaced mode. De noninterlaced mode of interlaced mode kan worden ingesteld, door middel van het instellen van de parameters van de grafische monitor. In de interlaced mode moet het eerste window een gelijk of groter aantal beeldlijnen hebben dan het aantal beeldlijnen van de grafische monitor, zodat het tweede window niet op het scherm komt.
- Het instellen van de beeldvergrotingsfactor op waardes van 1 tot 16 maal.
- Het projecteren van een tweetal demonstratie figuren op het scherm. Tevens kan er een programmaatje opgestart worden, waardoor het window zich automatisch over het gehele geheugen beweegt. Op deze wijze kan de inhoud van het gehele grafische geheugen geinspecteerd worden.

6.5 Werking Basis Software

Het programma kan in drie modes werken: De normal mode, de linestyle mode en de character mode. De mode waarin het systeem zich bevindt wordt weergegeven door de variabele isl.mode, dus in de internal state list. De normal mode wordt gebruikt bij het normale tekenen. De linestyle mode wordt gebruikt indien er stippellijnen worden getekend. De character mode tenslotte, wordt gebruikt indien er grafische characters op het beeldscherm afgebeeld moeten worden.
Iedere mode heeft zijn eigen variabelen. In hoofdstuk 4 is reeds uitgelegd dat de grafische kaart in de multipixel mode en in de niet-multipixel mode kan werken. In de multipixel mode krijgen de ingetekende pixels bij het tekenen allen dezelfde kleur index, onafhankelijk van wat de oude kleur van de pixels was. In de niet-multipixel mode wordt de nieuwe kleur index van een pixel afhankelijk van de oude kleur index van het pixel. De software is nu zodanig ingericht, dat de operator kan instellen of er in de multipixel

mode, of de niet-multipixel mode getekend wordt. Dit doet hij door multipixel aan of uit te zetten. Multipixel kan aan en uit worden gezet in alle drie de modes: De normal mode, de linestyle mode en de character mode en wel onafhankelijk van elkaar. Zo kan in de character mode multipixel aan worden gezet, in de normal mode multipixel uit en in de linestyle mode multipixel bijvoorbeeld weer aan. In de internal state list variabele isl.mp wordt aangegeven of multipixel aan of uit is in de normal-, character-, en linestyle mode. Isl.mp is een structure bestaande uit 3 booleans die de toestand van multipixel in de drie modes representeren. De drie modes, normal mode, linestyle mode en character mode, gebruiken ook ieder hun eigen modify tabel, bij het tekenen, in de terugkoppellus. De tabel die in de normal mode gebruikt wordt, kan door de operator vanaf het keyboard worden ingesteld. Dit tabelnummer wordt onthouden door de software, in de internal state list onder de variabele isl.tablenr. De tabellen die in de linestyle mode en in de character mode worden gebruikt, liggen softwarematig vast en zijn gedefinieerd in de file constant.h.
Wanneer er characters worden getekend, dan wordt er een rechthoekig veld gevuld met een achtergrond kleur en een voorgrond kleur van het character zelf. De achtergrond kleur wordt bepaald door modify tabel nummer 2, terwijl de voorgrond kleur bepaald wordt door modify tabel nummer 3. Als er een stippellijn wordt getekend, zal deze stippellijn uit twee kleuren bestaan. De ene kleur wordt bepaald door de modify tabel nummer 4. De tweede kleur wordt bepaald door de modify tabel nummer 5. Het tekenen van stippellijnen en characters gebeurt dus door het continu heen en weer schakelen tussen 2 tabellen. Er kan softwarematig echter slechts 1 tabel geselecteerd worden. Het schakelen tussen de 2 tabellen gebeurt dan als volgt:
Het nummer van de ene tabel (achtergrond kleur bij characterdrawing) is het nummer van de geselecteerde tabel, waarbij het laatste digit is gereset. Het nummer van de tweede tabel wordt dan het geselecteerde tabelnr, waarbij de laatste digit is geset. Bijvoorbeeld selecteer tabel 2, dan wordt in de normal mode alleen met tabel 2 getekend. In character- en linestyle mode wordt echter geschakeld tussen de tabellen 2 en 3.
Het patroon van de stippellijn die getekend wordt, wordt bepaald door het linestyle register te laden. Dit laden van het linestyleregister, kan door de operator vanaf het keyboard gebeuren. Er moet dan een 16 bits getal worden ingevoerd, door middel van het invoeren van vier hexadecimale waarden. Dit patroon van nullen en enen bepaalt dan het zich herhalend patroon van 16 bits van de stippellijn.
De tabellen 14 en 15 tenslotte, worden gebruikt bij het intekenen van de crosshair. De crosshair wordt ingetekend in het vierde bit plane. Het intekenen van de crosshair vindt plaats in de niet-multipixel mode. Bij het intekenen en uitvegen van de crosshair, worden respectievelijk de modify tabellen 15 en 14 gebruikt. Deze tabellen zijn zodanig geladen, dat bij het tekenen alleen de bits in het vierde bitplane geset (modify tabel 15) of gereset (modify tabel

14) worden. Hierbij blijven de bits uit de eerste drie bitplanes dus ongewijzigd. De crosshair staat dus in het vierde bitplane, de grafische tekening in de eerste drie bitplanes.
Verder zijn nog de moeite van het vermelden waard de functies in de file timecri0.c. In deze file staan de tijdkritische commando's. Als de GDC namelijk aan het tekenen is, mogen bepaalde commando's niet worden uitgevoerd, zoals het laden van modify tabellen, of het instellen van de character vergroting, of het laden van de colour look-up tabel etc.. Voordat dit soort commando's mogen worden uitgevoerd, moet eerst worden gewacht totdat de GDC klaar is met het tekenen. Hiertoe is een speciale functie waitgdc gedefinieerd in de file gdc0.c. Voordat een van de functies uit de file timecri0.c aangeroepen kan worden, moet dus eerst de functie waitgdc aangeroepen worden. Een andere mogelijkheid is, dat eerst een functie wordt aangeroepen, waarin de waitgdc functie is verwerkt, zoals bijvoorbeeld de functie fslnormal uit de file hardw0.c.

Het zal duidelijk zijn, dat het monitor programma dat de commando invoer verzorgt, niet een definitieve versie is. Het biedt alleen de mogelijkheid voor de systeem ontwerper, om het volledige grafische systeem in al zijn functies uit te testen. In de uiteindelijke vorm, waarin de software gecombineerd wordt met GKS, zullen de commmando verwerkings utilities uit niveau 2 dan ook vervallen. De interactie met de operator zal dan plaats moeten vinden via een applicatie programma van GKS.
De gedetailleerde werking van het programma kan het beste worden bestudeerd aan de hand van de software listings. In deze listings is zeer de nadruk gelegd op een overzichtelijke uitvoering met een dusdanige hoeveelheid commentaar, dat het programma alleen aan de hand van de software listings bestudeerd kan worden. Hierbij zijn ook de listings van alle gedefinieerde procedures met bijbehorende filenaam en regelnummer, benevens de listing van alle define statements op alfabetische volgorde, een belangrijk hulpmiddel.

7 Werking GKS

7.1 Lokalisatie en Herkomst van de Sources

De GKS versie die geimplementeerd zal worden op het Unix systeem, is een versie die ontwikkeld is door het Centrum voor Wiskunde en Informatica te Amsterdam. De sources van deze GKS implementatie zijn aangekocht door het rekencentrum onder verantwoordelijkheid van Rens Kessner. De sources van de GKS implementatie zijn op de Unix Vax gezet onder de directory /usr/users/rcrens/cwigks. Onder deze directory staan alle GKS sources inclusief de subdirectories. Deze gehele directory met bijbehorende subdirectories is gecopieerd naar de directory /usr/users/elebst3/bert/gks. De sources zijn als volgt georganiseerd: in de directory gks bevinden zich alle files die benodigd zijn voor de GKS implementatie benevens 6 subdirectories. Deze subdirectories zijn:

- Doc: In deze directory bevindt zich alle documentatie over de GKS implementatie.
- Fortran: Hierin bevindt zich de bijbehorende fortran schil.
- Lint: Hierin bevinden zich enkele files die door de lint syntax checker gebruikt kunnen worden, bij het checken van de syntax van een GKS applicatie programma.
- Mains: Hierin bevinden zich voornamelijk testprogramma's. Dit zijn GKS applicaties die verschillende eenvoudige figuren op het scherm brengen, waarmee GKS uitgetest kan worden.
- VMS: Deze directory bevat een programma om de GKS files op een VMS Vax in te lezen.
- bin: In deze directory bevinden zich enkele shell procedures die dienen om verschillende stukken van GKS te printen, en tevens enkele sed (stream editor) utilities.

7.2 Listings

De GKS implementatie bestaat uit meer dan 200 files. Deze files staan in de hierboven genoemde directory gks. Bij de bestudering van de GKS implementatie was het nodig om al deze files te listen. Het listen van alle 200 files achtereenvolgens in alfabetische volgorde, zou echter een niet hanteerbaar groot pak papier opleveren. Er is daarom gekozen voor een methode, waarbij de files in logische blokken zijn onderverdeeld en in gedeeltes zijn gelist. Bij de vermelding van de geliste files is gebruik gemaakt van de ster operator. De ster operator wordt toegepast in filenamen. Wanneer we bijvoorbeeld het Unix commando pr a* geven, dan zal Unix alle files gaan printen die met een a beginnen. De ster operator is hieronder op analoge wijze gebruikt. Zo moet s_* gelezen worden als zijnde alle files die beginnen met s_. De GKS files uit de directory gks zijn in de volgende blokken gelist (geprint):

- s_* files: Alle "set" files.

- u_* files: De "utility" files.

- De driver files, waaronder tevens begrepen de metafile (driver) files:
  t40*
  tek*
  foo*
  mf*
  mi*
  ver*.c
  ver*.h
  Hiervan zijn mf* en mi* metafile (driver) files. De andere files zijn workstation drivers.

- aed* files. Deze files zijn de driver files behorende bij het aed512 grafisch werkstation. De files zijn apart gelist, aangezien deze driver gebruikt is als uitgangspunt, bij het ontwikkelen van de driver voor de GEB grafische kaart.

- Van de resterende files zijn de c files gelist. D.w.z. de files met achtervoegsel .c.

- Van de resterende files zijn eveneens de header files gelist: De files met achtervoegsel .h.

- Tenslotte zijn alle overige files gelist.

Verder zijn uit de sub directories nog een aantal files gelist en wel de volgende:

- DOC/* files. De bij GKS behorende on line documentatie files. De listing van deze files is echter gelijk aan de, bij de GKS implementatie behorende, documentatie set van twee paketten stencils op A4 formaat (zie lit. 3 en 4). De files staan in nroff formaat.

- Mains/PICTURES. Bevat informatie over de aanwezige GKS applicatie testprogramma's.
  VMS/* files. Alle files uit de directory VMS.
  bin/* files. " " " " " bin.

- Tenslotte is nog een listing gemaakt van alle procedures die in GKS gedefinieerd zijn, met de filenaam waar de definitie heeft plaatsgevonden en het regelnummer in de file.
  Evenzo is een listing gemaakt van alle #define statements in de GKS files, voorafgegaan door de bijbehorende filenaam en het regelnummer.
  Beide listings zijn gemaakt met behulp van twee zelf ontwikkelde utilities, nl. prprocs en prdefs. Zie hiervoor hoofdstuk 9.3 Utilities.

Overigens zijn de bovengenoemde listings i.v.m. de te grote

omvang niet bij het verslag gevoegd.

7.3 Globale Werking GKS

De globale werking van GKS zal worden verduidelijkt aan de hand van fig. 7.1.

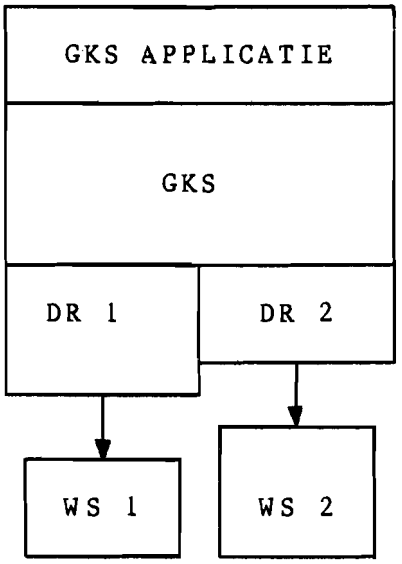

FIG 7.1 GKS applicatie met twee werkstations

We zien in de figuur vier software modules die met elkaar tot een module gelinkt en geladen zijn: GKS APPLICATIE, GKS, DR 1 (Driver 1) en DR 2 (Driver 2). Verder zijn er twee werkstations aanwezig: WS 1 en WS 2. Een GKS werkstation is een grafische unit. Dit kan zijn een grafisch beeldscherm, of ook bijvoorbeeld een plotter. Eventueel kan er ook nog een keyboard voor input bij zitten. De GKS APPLICATIE module is een C programma dat onder meer bestaat uit GKS functie aanroepen. Een voorbeeld van zo'n programma is weergegeven in hoofdstuk 7.4. De laag GKS bestaat onder meer uit een aantal statelists welke in hoofdstuk 7.5 nader behandeld zullen worden en uit gestandariseerde GKS functies, welke door de GKS applicatie worden aangeroepen. De GKS standaard is in feite namelijk alleen een definitie van gestandariseerde grafische functies. De GKS implementatie van het C.W.I. bestaat dan ook uit een library (bibliotheek), waarin alle GKS functies aanwezig zijn. Indien een GKS applicatie gecompileerd wordt, dan zal de compiler automatisch die GKS functies uit de library halen, die door de GKS applicatie worden aangeroepen.
Met de GKS functies zijn tevens twee drivers meegelinkt. Een driver verzorgt de DI/DD (Device Independent/Device Dependent) interface. De verschillende werkstations die door GKS worden ondersteund, zullen namelijk in het algemeen verschillend moeten worden aangestuurd. De werkstations zijn namelijk meer of minder intelligent en kunnen ook volledig

verschillende functies hebben.
De interface tussen de GKS APPLICATIE en de GKS module is geheel gestandariseerd. Deze is namelijk vastgelegd in de GKS standaard. De interface tussen GKS en de drivers is nog niet gestandariseerd. Deze interface moet worden vastgelegd door de ontwikkelaars van een GKS implementatie. De standarisatie van deze interface is echter wel in ontwikkeling. Zoals blijkt uit figuur 7.1 ligt de interface tussen GKS en de verschillende drivers op een en hetzelfde niveau. Alle drivers worden met dezelfde commando's aangesproken. De functie van de driver is het omzetten van de commando's die ontvangen worden van GKS, in commando's die door de werkstations kunnen worden uitgevoerd. Het is duidelijk dat een minder intelligent werkstation, zoals in de figuur werkstation 1, een uitgebreidere driver nodig zal hebben dan een intelligenter werkstation zoals werkstation 2.
In de GKS implementatie van het C.W.I. ligt de interface tussen GKS en de drivers op een vrij laag niveau. Dit heeft het volgende voordeel: Indien er voor een bepaald type werkstation nog geen standaard driver voorhanden is, dan zal deze driver relatief eenvoudig zelf kunnen worden ontwikkeld. Het nadeel is echter dat men van zeer intelligente werkstations niet alle mogelijkheden zal kunnen benutten, omdat de interface tussen GKS en de driver op een te laag niveau zou kunnen staan. Het C.W.I. is echter bezig om nog een tweede interface niveau tussen GKS en de drivers vast te leggen. Deze tweede interface, die via conditionele compilatie geselecteerd kan worden, ligt op een hoger niveau en kan zodoende beter gebruik maken van zeer intelligente werkstations.
Een GKS applicatie die van meerdere werkstations gebruik maakt zal als volgt functioneren: Indien een bepaald werkstation in gebruik gesteld moet worden, moet eerst de GKS open workstation functie aangeroepen worden. Hierdoor wordt er voor dit werkstation een workstation statelist gecreeerd, waarmee het werkstation aan GKS bekend wordt. Pas als we echter de activate workstation functie aangeroepen hebben, zal de grafische informatie daadwerkelijk op het werkstation worden afgebeeld. GKS werkt zodanig, dat als er getekend wordt, de grafische informatie naar alle geactiveerde werkstations gestuurd wordt.

7.4 GKS Applicatie Programma

We zullen nu de werking van GKS nader gaan bestuderen. GKS is een standaard voor twee dimensionale grafische toepassingen. Met GKS is het bijvoorbeeld mogelijk, om figuren te tekenen en tekst te tekenen in allerlei lettertypen. Eerder gedefinieerde figuren kunnen worden opgeroepen en worden onderworpen aan gecompliceerde coordinaten transformaties. De GKS standaard is echter dermate veel omvattend, dat het niet mogelijk is om deze standaard in dit verslag te behandelen. Enige kennis van GKS zal dan ook bij de lezer aanwezig worden geacht. Een beschrijving van de GKS standaard kan worden gevonden in literatuur 2).

De werking van de GKS software zal in de volgende hoofdstukken behandeld worden aan de hand van een klein applicatie programma. De listing van dit programma volgt hieronder.

```
 1   /*******begin van test programma**********************/
 2   #include "cgksincl.h"
 3
 4   Gks  *gkss;
 5   Wss  *wss_geb;
 6   Wc   p[2];
 7
 8   main()
 9   {
10        gkss =open_gks(NULL, NULL, NULL, "errormessages");
11        wss_geb =open_ws(stdin, stdout, geb001);
12        activate(wss_geb);
13
14        p[0].w_x = p[0].w_y = 0.0;
15        p[1].w_x = p[1].w_y = 1.0;
16        polyline(2,p);
17
18        deactivate(wss_geb);
19        close_ws(wss_geb);
20        close_gks();
21   }
22
23   /*********einde van test programma*********************/
```

Als dit programma test.c zou heten, dan zou het gecompileerd kunnen worden tot een executable object file onder Unix, door het commando cc test.c -lgks -lm. Hierdoor wordt de test.c file gecompileerd, waarbij de compiler de standaard gks library (-lgks) en de mathematische library (-lm) doorzoekt en hieruit de aangeroepen functies meelinkt.
In regel 2 van het programma vinden we de statement #include "cgksincl.h". Hiermee wordt een file geinclude, die in alle applicatie programma's aanwezig moet zijn. Deze file bestaat zelf uit een aantal opeenvolgende include statements, zodat er door deze file te includen, een dertiental header files in het applicatie programma worden geinclude. In deze header files staan o.a. definities van constanten, definities van macro's, type definities, de definities van de standaard Unix input en output routines etc..
Op de vierde regel zien wij de declaratie van de pointer gkss welke wijst naar een structure van het type Gks: de Gks statelist. In regel 5 wordt de pointer wss_geb gedefinieerd. Deze pointer wijst naar een structure van het type Wss: een workstation statelist. Deze twee statelists zijn zeer essentieel binnen GKS en zullen daarom hieronder behandeld worden.

7.5 GKS Statelists

De type definitie van de Gks en Wss statelists is aanwezig

in de file cgks.h die geinclude wordt in de file cgksincl.h. Hieronder volgt de listing van de beide statelists. Deze listing is vrijwel geheel letterlijk overgenomen uit de file cgks.h.

```
typedef struct  Gks  {
     Ntran      **gk_ntran;
     Ntran      *gk_curnt;

     Asflags    gk_asf;

     Bindex     gk_line;
     Ltype      gk_lrlt;
     Lwidth     gk_lrlw;
     Cindex     gk_lrci;

     Bindex     gk_mark;
     Mtype      gk_mrmt;
     Msize      gk_mrms;
     Cindex     gk_mrci;

     Bindex     gk_text;
     Fofr       gk_txfp;
     Charef     gk_chef;
     Charsp     gk_chsp;
     Cindex     gk_txci;
     Charht     gk_chht;
     Wc         gk_chup;
     Path       gk_path;
     Talign     gk_txal;

     Bindex     gk_area;
     Istyle     gk_fais;
     Sindex     gk_fasi;
     Cindex     gk_faci;

     Wc         gk_patsiz;
     Wc         gk_patref;
     Wss        **gk_opws;
     Wss        **gk_actv;
     Seg        *gk_opsg;
     Seginst    *gk_segs;

     Pickid     gk_pikid;
     Bool       gk_clip;
     Bool       gk_more;
     } Gks;


typedef  struct  Wss  {
     Wsd        *ws_wsd;
     Wsis       *ws_wsis;

     Int        ws_id; /* workstation identifier */

     File       *ws_ifile; /* connection id; infile */
```

```
File        *ws_ofile; /* connection id; outfile */

Bool        ws_actv;
Seginst     *ws_segs;

Defmode     ws_defr;  /* deferral mode */
Bool        ws_waitr; /* implicit reg; SUPPRESSED=TRUE */
Bool        ws_empty; /* display surface empty */
Bool        ws_newfr; /* new frame action necess.
                         at update */
Bool        ws_trupd; /* ws transf update st;
                         NOTPENDING=FALSE */
Nrect       ws_reqw;
Nrect       ws_curw;
Drect       ws_reqv;
Drect       ws_curv;

Int         ws_nline;
LineRep     *ws_line;
Int         ws_nmark;
MarkRep     *ws_mark;
Int         ws_ntext;
TextRep     *ws_text;
Int         ws_narea;
AreaRep     *ws_area;
Int         ws_npatt;
PattRep     *ws_patt;

Int         ws_ncolr;
Colour      *ws_colr;
Idevice     *ws_devs[6];
} Wss;
```

We zien dat alle elementen uit de structure van het type Gks beginnen met gk_, terwijl alle elementen uit de structure van het type Wss beginnen met ws_. De hierboven geliste type definities definieren alleen hoe een structure van het type Gks of wel Wss er uit moet zien. Zij reserveren echter geen geheugenruimte voor deze structuren. Het zijn dus geen declaraties. Alle statelists die door GKS gebruikt worden komen dynamisch tot stand. Dat wil zeggen dat, tijdens de uitvoering van GKS onder Unix, GKS zelf geheugenruimte gaat reserveren voor alle statelists. Dit gebeurt door middel van de Unix system calls malloc en calloc. Er is slechts een statelist van het type Gks. Er zijn echter meerdere statelists van het type Wss. Voor ieder werkstation dat geopend wordt, wordt er namelijk een Wss statelist aangebracht.

Enkele Elementen uit de Gks Statelist

We zullen nu enkele elementen uit de Gks statelist bespreken. Het eerste element gk_ntran is een pointer naar een lijst van pointers, die ieder naar een normalisatie transformatie toewijzen. Het element gk_curnt is een pointer, die wijst naar de current normalisatie transformatie. Dat wil zeggen de normalisatie transformatie die momenteel in ge-

bruik is. Als een van de laatste elementen zien we onder meer het element gk_opws. Dit is een pointer naar een lijst van pointers, waarvan iedere pointer wijst naar een workstation statelist van een werkstation dat geopend is. Het volgende element gk_actv is een pointer wijzend naar een lijst van pointers, die ieder wijzen naar de workstation state list van een active workstation. D.w.z. een werkstation dat eerst geopend is en daarna geactiveerd is. Het element gk_opsg is een pointer die wijst naar het geopende segment. Een segment is een mechanisme om een tekening in op te slaan. Er mag maar een segment tegelijkertijd open zijn.

Unbundled Attributen

Verder staan er in de Gks state list nog een groot aantal attributen. De attributen bepalen hoe een tekening er uit komt te zien. Voorbeelden van attributen zijn het lijn type, dat gestippeld of doorgetrokken kan zijn, de lijn dikte, de kleur van de lijn. Het blok dat begint met gk_line geeft informatie over de lijn attributen. Het blok dat begint met gk_mark geeft informatie over de marker attributen. Markers in GKS kunnen o.a. zijn een ., een *, een +, een x. De markers worden gebruikt om in tekeningen bepaalde punten te markeren. Het blok dat begint met gk_text bevat de attributen die worden gebruikt bij het tekenen van tekst. Het blok dat begint met gk_area bevat de attributen betreffende het fill area mechanisme. Hiermee kan een bepaald gebied gevuld worden, d.w.z. geheel ingekleurd of bijvoorbeeld schuin gearceerd.
Indien we bijvoorbeeld een lijn willen gaan tekenen, dan moeten eerst de volgende attributen worden ingesteld: Het lijn type, de lijn dikte en de kleur van de te tekenen lijn. Deze 3 attributen worden opgeslagen in de respectievelijke elementen gk_lrlt van het type Ltype (Linetype), gk_lrlw van het type Lwidth (Linewidth) en in gk_lrci van het type Cindex (Colour index). Het instellen van deze attributen moet door speciale GKS functie calls gebeuren. GKS zorgt er dan zelf voor, dat de elementen uit de GKS statelists op de juiste waarde worden ingesteld.
De GKS standaard heeft ook een manier gedefinieerd, waarmee de waarden van de attributen door de gebruiker kunnen worden opgevraagd. Dit gebeurt door middel van de zogenaamde inquiry functies. De implementatie van deze inquiry functies in de GKS implementatie van het C.W.I., gebeurt op een zeer eenvoudige wijze. Een inquiry functie is als een hele simpele macro gedefinieerd, waarmee eenvoudigweg het gevraagde attribuut uit de Gks statelist wordt gehaald.
Wanneer nu de attributen van de te tekenen lijn eenmaal ingesteld zijn, dan zullen alle volgende lijnen op deze manier getekend worden, d.w.z. met deze attributen. Evenals de lijn attributen kunnen ook de marker attributen, de tekst attributen etc. worden ingesteld. Hiermee zal dan vastliggen hoe de eventueel te tekenen markers en de af te beelden tekst zullen worden getekend.
Als we op deze manier te werk gaan, spreken we van globale ofwel unbundled attributen. Indien we op meerdere werkstations tegelijkertijd zouden tekenen, dan zouden met

unbundled attributen de tekeningen op alle werkstations gelijk worden.

Bundled Attributen

Het alternatief is het werken met bundled ofwel werkstation afhankelijke attributen. We moeten dan per werkstation gaan definieren hoe de lijnen, markers en tekst etc. moeten worden afgebeeld. Voor het afbeelden van bijvoorbeeld een lijn worden dan niet meer de attributen uit de Gks statelist gebruikt, maar de attributen uit de workstation statelist, van het werkstation waarop getekend wordt. In de type definitie van de workstation statelist zien we bijvoorbeeld het element ws_nline. Dit element geeft aan hoeveel verschillende soorten lijnen er op het betreffende werkstation kunnen worden afgebeeld. Het element ws_line is een pointer die wijst naar een structure van het type LineRep (Line Representation). Zo zal ws_line[0] een structure zijn waarin de eerste lijn representatie is opgeslagen, terwijl ws_line[1] de structure is waarin de tweede lijn representatie is opgeslagen. Het aantal verschillende lijn representaties wordt dus gegeven door ws_nline. Een lijn representatie heeft dezelfde elementen als welke we vinden in de Gks statelist, namelijk het lijn type, de lijn breedte, de colour index en de line index. De line index zal hieronder nog nader behandeld worden. Om een bepaalde lijn representatie op een werkstation in te stellen moeten we weer een speciale GKS functie aanroepen. Analoog aan de lijn representaties kunnen ook de marker representaties en de tekst representaties etc. worden ingesteld.
Wanneer we met GKS in de bundled mode lijnen of andere figuren tekenen, dan zullen die in het algemeen op de verschillende werkstations op een andere manier worden afgebeeld. De wijze waarop een lijn op een bepaald werkstation zal worden afgebeeld, is namelijk niet meer afhankelijk van de attributen in de Gks statelist, maar van de attributen die staan in de, bij ieder werkstation behorende, eigen workstation statelist. Het enige attribuut dat uit de Gks statelist wordt genomen is, in het geval van het tekenen van lijnen, het element gk_line van het type Bindex (Bundle index). Dit element geeft het nummer aan van de geselecteerde lijn representatie: De bundle index van de lijn. Indien gk_line bijvoorbeeld 2 is, dan zal uit de workstation statelists steeds die lijn representatie worden gebruikt, waarvan de bundle index 2 is. Deze lijn representatie structure bevat de bundle index, het lijn type, de lijn breedte en de kleur index van de lijn, zoals die op het werkstation moeten worden afgebeeld. De bundle index van de geselecteerde lijn wordt de line index genoemd. De line index kan worden ingesteld door middel van een GKS functie aanroep, waarmee de variabele gk_line in de Gks statelist op een bepaalde waarde wordt geset.
Nu is het echter zo, dat niet alle attributen tegelijkertijd in de bundled of unbundled mode hoeven te staan. Voor ieder attribuut afzonderlijk, kan worden ingesteld of het in de bundled of unbundled mode staat. Zo kan bijvoorbeeld, bij het tekenen van een lijn, het lijn type unbundled zijn, de

lijn breedte bundled en de kleur index van de lijn weer unbundled. Hierdoor wordt het lijn type attribuut uit de Gks statelist gehaald, de lijn breedte uit de afzonderlijke workstation statelists en de kleur index weer uit de Gks statelist. GKS houdt zelf bij voor de afzonderlijke attributen, of zij in de unbundled of wel in de bundled mode staan. Deze informatie bevindt zich in het element gk_asf van het type Asflags (Aspect source flags).

Overige Elementen uit de Workstation Statelist

In de workstation statelist staat eveneens de colour look-up tabel. Het element ws_ncolr geeft aan hoeveel verschillende kleur indexen de tabel heeft. Het element ws_colr is een pointer die point naar het begin van de colour look-up tabel. Ieder element uit deze tabel is een structure welke bestaat uit drie getallen tussen 0 en 1. Het eerste getal is een maat voor het aandeel van de kleur rood in de kleur index, het tweede getal geeft het aandeel van de kleur blauw in de kleur index aan, terwijl het derde getal het aandeel van de kleur groen aangeeft. Het eerste element uit de colour look-up tabel is de kleur index 0. De volgende elementen hebben een kleur index die steeds een waarde hoger is. Deze colour look-up tabel is een GKS colour look-up tabel.
In het werkstation zelf zal zich in het algemeen ook een colour look-up tabel bevinden. Op de huidige grafische kaart uit het Unix systeem is zo'n colour look-up tabel aanwezig. Hierin kunnen echter de kleuren rood, groen en blauw slechts of wel geheel aan, of wel geheel uit zijn. In totaal kunnen er dus slechts acht verschillende kleuren gemaakt worden. De informatie uit de GKS colour look-up tabel in de workstation statelist, moet worden overgebracht naar de colour look-up tabel op de grafische kaart. Dit moet zodanig gebeuren, dat de waarden in de colour look-up tabel op de grafische kaart, een zo goed mogelijke benadering zijn van de GKS colour look-up tabel in de workstation statelist.
We zien verder nog het element ws_wsd: een pointer naar een workstation description table van het type Wsd. Een workstation description table is een tabel die informatie bevat over een bepaald type werkstation. De workstation description tables worden door GKS, voor alle bij GKS bekende typen werkstations, dynamisch gecreeerd en daarna geladen vanuit de file wsdcap. In deze file staan, in gecodeerd formaat, de kenmerken van alle aanwezige werkstations beschreven.
Indien er meerdere werkstations van hetzelfde type aanwezig zijn, dan is er voor deze werkstations tesamen precies een workstation description table. Er kunnen echter wel meerdere workstation statelists aanwezig zijn. Namelijk voor ieder geopend werkstation een. Indien de geopende werkstations van hetzelfde type zijn, dan zullen de pointers ws_wsd uit de workstation statelists allen naar dezelfde workstation description table wijzen.
De pointer ws_wsis point naar een interne workstation statelist welke door GKS alleen voor intern gebruik gebruikt wordt.
Tenslotte zien we nog de elementen ws_ifile en ws_ofile.

Deze twee elementen zijn pointers naar respectievelijk de input file en output file. De output file is de file waarnaar GKS output moet plegen, om output naar het betreffende werkstation te plegen. Indien GKS input wil plegen vanuit het werkstation, moet het de infile lezen. Deze twee pointers geven GKS dus de kennis van de plaats, waarop het werkstation aan het Unix systeem is aangesloten.

7.6 Open GKS en Open Workstation

Verwijzend naar het testprogramma uit hoofdatuk 7.4 zien we op regel 10 als eerste statement de aanroep van de functie open_gks. De eerste drie parameters van deze functie worden door de software niet geinterpreteerd en kunnen dus NULL genomen worden. De vierde parameter is de naam van een file waar de error messages naar toe moeten worden gestuurd. De open_gks functie verzorgt de dynamische toewijzing van geheugenruimte voor de verschillende statelists en alle workstation description tables en zorgt voor de initialisatie van deze statelists. De workstation description tables worden ingelezen vanuit de file wsdcap. De functie returnt een pointer naar de Gks statelist. Met behulp van deze pointer kan het applicatie programma de Gks statelist dus bereiken.
In regel 11 wordt als tweede statement de open_ws functie aangeroepen. Met deze functie wordt een werkstation geopend. De eerste parameter is de naam van de file die GKS moet lezen om input op te vragen van het werkstation. De tweede parameter is de file waarnaar GKS moet schrijven om output te plegen naar het werkstation. De derde parameter is het adres van een functie. Deze functie moet gedefinieerd zijn in de driver module, die bij het te openen werkstation behoort. De naam van deze functie zal in het algemeen overeenkomen met de naam van het te openen werkstation. De functie zorgt ervoor, dat de juiste driver wordt geinstalleerd, en dat GKS de driver functies kan aanroepen. De compiler herkent geb001 als zijnde het adres van een functie, omdat in de file cgkswstypes.h de functie definitie int geb001(); moet staan. Geb001 staat in dit geval voor het grafisch werkstation ontwikkeld op de vakgroep EB, waarvan het volgnummer 001 is. De naam aanduiding is gekozen in analogie met de naam van het AED512 werkstation. Dit is een grafisch raster scan werkstation, waarvan de driver is gebruikt bij de ontwikkeling van de GEB001 driver.
In de driver van het GEB001 werkstation bevindt zich dus de functie geb001(). Om aan te geven hoe, in de open_ws functie, de geb001 functie aangeroepen wordt, laten we hieronder een klein stukje van de definitie van de open_ws functie volgen.

```
1     Wss *
2     open_ws(ifile, ofile, wstype)
3     File   *ifile, *ofile; /* connection id's */
4     int    (* wstype)();
5     {
6          ....
```

```
7           .... (*wstype)() ....
8           ....
9     }
```

We zien in de vierde regel dat de derde formele parameter wstype gedefinieerd wordt, als zijnde een pointer naar een functie die een integer returnt. In regel 7 wordt deze functie aangeroepen. Aangezien het functie adres wstype bij de aanroep van de open_ws functie als parameter wordt meegegeven, zal de module waarin de functie (*wstype)() is gedefinieerd, door de compiler worden meegeladen. Welnu deze functie is in de driver module gedefinieerd, waardoor automatisch de juiste driver wordt meegeladen.
De functie geb001(), die op deze manier wordt aangeroepen, zorgt er tevens voor, dat de screen pointer in de workstation description table van dit werkstation geinstalleerd wordt. De functie van de screen pointer zal later nog aan de orde komen.
Verder wordt door de open_ws functie de workstation statelist dynamisch gecreeerd, en geladen vanuit de bijpassende workstation description table. Vervolgens wordt de workstation statelist verder geinitialiseerd. De open_ws functie returnt uiteindelijk de pointer naar de, bij het geopende werkstation behorende, workstation statelist. Deze pointer moet als parameter worden meegegeven bij functies zoals activate (zie test programma in hoofdstuk 7.4 regel 12), deactivate (regel 18) en close_ws (regel 19). Indien er bijvoorbeeld twee werkstations geopend moeten worden, dan zal de open_ws functie twee keer moeten worden aangeroepen. Er zullen dan ook twee verschillende workstation statelist pointers worden gereturnd: Voor elk werkstation een. Als nu bijvoorbeeld de GKS functie activate wordt aangeroepen, met als parameter de pointer naar de workstation statelist van het eerste werkstation, dan zal dat eerste werkstation geactiveerd worden. Om het tweede werkstation te activeren, moet als parameter de pointer naar de workstation statelist van het tweede werkstation meegegeven worden.
In regel 16 zien we de GKS functie polyline. Aan de GKS polyline functie moet als tweede parameter worden doorgegeven een array met punten (uitgedrukt in x en y coordinaten), en als eerste parameter een getal dat het aantal punten uit de array aangeeft. De polyline functie zal de achtereenvolgende punten dan met lijnstukken doorverbinden. Deze polyline zal op alle geactiveerde werkstations (in dit geval is dat er maar een) zichtbaar worden. In het gegeven voorbeeld zitten er maar twee punten in de array p, zodat de polyline zal bestaan uit een lijnstuk.
In regel 18 wordt het werkstation gedeactiveerd. In regel 19 wordt het werkstation gesloten en in regel 20 wordt GKS gesloten.
Reeds eerder is de screen pointer ter sprake gekomen. Deze pointer wordt, door de geb001 functie uit de driver module, geinstalleerd in de workstation description table. Deze pointer is essentieel bij de overdracht van informatie van GKS naar de driver. We zullen nu gaan behandelen hoe GKS

commando's geeft aan de verschillende drivers.

7.7 GKS/Driver Interface

In de file cgks.h vinden we onder meer de type definitie van de structure Screen. Deze structure, ook wel genaamd het driver field, legt de interface vast tussen GKS en de drivers. De type definitie van deze structure is hieronder weergegeven.

```
/* Type definitie van het driver field */
typedef struct {
     Ic         s_size;
     Ic         s_chsz;
     Int        s_chht;
     Ic         s_hit;
     int        s_code;
     Cindex     s_ccol;
     char       s_flag;
     char       s_timo;
     Bool       (*s_tera)(); /* open ws; total erase */
     Bool       (*s_eral)(); /* set line erase mode */
     Bool       (*s_clos)(); /* close workstation */
     Bool       (*s_clpr)(); /* set clip rectangle */
     Bool       (*s_msge)(); /* send a message to screen */

     Bool       (*s_line)(); /* linepieces */
     Bool       (*s_mark)(); /* markers */
     Bool       (*s_char)(); /* hard characters */
     Bool       (*s_pxst)(); /* pixels (all pixs same or
                                different) */
     Bool       (*s_runl)(); /* pixels via a run length
                                encoding */
     Bool       (*s_cell)(); /* directly cellarray to dev. */
     Bool       (*s_gdpr)(); /* gen draw prim */
     Bool       (*s_rect)(); /* use it as rect erase for
                                teklike term */

     Bool       (*s_latt)(); /* set polyline attributes */
     Bool       (*s_matt)(); /* set polymarker attributes */
     Bool       (*s_tatt)(); /* set text attributes */
     Bool       (*s_fatt)(); /* set fillarea attributes */
     Bool       (*s_colr)(); /* select colour */
     Bool       (*s_colt)(); /* set colour table */
     } Screen;
```

We zien eerst een aantal elementen van de structure die beginnen met s_. Dit zijn elementen die gegevens bevatten van het betreffende werkstation, zoals de grootte van het scherm, de hoogte van de afgebeelde characters en de momenteel ingestelde kleur index op het werkstation. Daaronder volgt een hele reeks elementen welke als volgt gedefinieerd zijn:

```
Bool       (*s_xxxx)();
```

Hiermede wordt een pointer s_xxxx gedefinieerd, die point naar een functie, welke een waarde returnt van het type Bool. Deze pointer s_xxxx kan ook worden opgevat, als het adres van een functie. Alle functie adressen die op deze manier gedefinieerd zijn, zijn adressen van functies welke in de driver aanwezig moeten zijn, en door GKS kunnen worden aangeroepen. Er moet dan ook in elke driver module een declaratie aanwezig zijn, van een structure van het type Screen. Hieronder volgt de declaratie en initialisatie van deze structure, zoals gedefinieerd in de GEB001 driver in de file geb001.c. Er moet hier wel worden opgemerkt, dat de hier weergegeven structures van het type Screen een enigszins vereenvoudigde weergave van de werkelijkheid zijn. De functies die input verzorgen van het werkstation zijn namelijk weggelaten. In de huidige implementatie van de GEB001 driver zijn deze input functies niet gedefinieerd.

```
/* Declaratie en initialisatie van het driver field van de
 * GEB driver
 */
Screen  g_geb001[1] = {
     MAXX, MAXY,     /* Main screen size */
     CHARX, CHARY,   /* Hard character size */
     CHARHT,         /* Hard character height */
     0, 0,           /* Hit location */
     0,              /* s_code: Hit code */
     0,              /* Colour */
     S_CHANGED,      /* s_flag ; status */
     '\0',           /* s_timo */
     geberase,       /* Total erase ; open ws */
     NULL,           /* No erase mode; use colour 0 */
     final,          /* close ws */
     NULL,           /* set clipping rectangle */
     gebmessage,     /* message from the GKS message fun */

     drawline,       /* Line drawing */
     drawmark,       /* Marker */
     drawhard,       /* Hard characters */
     gebpixelstream,/* Write horiz. run; either 1 or
                       different colors */
     gebrle,         /* run length encoding */
     cells,          /* direct cellarray pipe */
     gendrawpr,      /* general drawing primitive */
     rectan,         /* No seperate rectangle erase;
                       use colour 0 */

     setlatt,        /* Set polyline attributes */
     setmatt,        /* Set polymark attributes */
     settatt,        /* Set text attributes */
     setfatt,        /* Set fill area attributes */
     setcol,         /* Select Colour */
     setcolt,        /* Set Colour Table */
     } ;
```

Met de regel Screen g_geb001[1] wordt een array van

structures gedefinieerd. Deze array bevat echter slechts een element, namelijk de driver field structure g_geb001[0]. Door te definieren "Screen g_geb001[1]" in plaats van "Screen g_geb001" wordt bereikt, dat g_geb001 het adres is van het eerste element van het driver field. Een gelijkwaardige definitie zou zijn geweest: "Screen g_geb001[] = {... . Achter het = teken wordt het driver field geinitialiseerd. Eerst worden er een aantal constanten ingevuld, die o.a. de beeldscherm afmetingen aangeven, de afmetingen van een character zoals dat op het scherm standaard wordt afgebeeld, de momenteel ingestelde kleur index en enkele andere variabelen. Hierna volgen, met als eerste geberase, de adressen van de driver functies zoals deze in de driver module zijn gedefinieerd. In de driver module zijn dus o.a. de volgende functies gedefinieerd: geberase(), final(), gebmessage(), drawline() etc.. De functienamen NULL geven aan, dat de betreffende functie in de driver niet geimplementeerd is.
Hoe kan GKS nu de driver functies zoals drawline en drawhard in werking stellen? Drawline zorgt ervoor dat er een polyline op het grafisch werkstation wordt getekend, terwijl drawhard een string van characters afbeeldt op het beeldscherm. GKS zal hiervoor de volgende functie aanroepen plegen:

```
screenpntr->s_line(wss_geb, n_points, array_points,
                   line_type, line_width);
screenpntr->s_char(wss_geb, string_pointer, position);
```

Screenpntr is de pointer naar het Screen driver field dat in de driver module gedefinieerd is.
Indien we een polyline willen tekenen, moeten we uit het driver field het element s_line adresseren. Dit element heeft in het geb driver field als waarde het adres van de drawline functie. Indien we een hard character op het grafisch werkstation willen afbeelden, moeten we uit het driver field het element s_char selecteren. Zie de type definitie van de screen structure.
Indien de pointer screenpntr point naar het driver field g_geb001 zoals dit is gedeclareerd in de GEB001 driver module (zie de listing hierboven van deze structure), dan zullen de bovenstaande GKS calls dus het volgende tot gevolg hebben: screenpntr->s_line zal de drawline functie aanroepen en screenpntr->s_char zal de drawhard functie uit de GEB001 driver aanroepen.
We zien dat in beide functie aanroepen als parameter de variabele wss_geb wordt meegegeven. Dit moet de pointer zijn naar de workstation statelist van het betreffende werkstation. De driver functies moeten deze workstation statelist namelijk kunnen bereiken om ernaar output te kunnen plegen, en om er gegevens vandaan te kunnen halen. Twee elementen uit de workstation statelist worden door iedere driver functie geraadpleegd. Namelijk de file pointers ws_ifile en ws_ofile, die aangeven onder welke files het werkstation beschikbaar is voor respectievelijk input en output. Alle driver functies doen in- en output van/naar de files die aangewezen worden door de file pointers wss_geb->ws_ifile en wss_geb->ws_ofile.

Het is bijvoorbeeld heel goed mogelijk dat twee werkstations van hetzelfde type, op verschillende plaatsen op het Unix systeem zijn aangesloten. Aangezien de werkstations van hetzelfde type zijn, worden ook dezelfde driver functies aangeroepen (afkomstig uit dezelfde driver field structure), voor in- en output naar de werkstations. Alleen door de workstation statelist pointer wss_geb, die als parameter moet worden doorgegeven, kan dan nog onderscheid gemaakt worden tussen de beide werkstations. Aangezien de workstation statelist dynamisch is gecreeerd door GKS, kunnen de driver functies zelf geen weet hebben van de plaats waar de workstation statelist zich bevindt. De pointer naar deze workstation statelist moet dus door GKS aan de driver functies als parameter worden meegegeven.
Het soort en het aantal parameters die bij de verschillende driver functie aanroepen moeten worden meegegeven, liggen niet vast in de definitie van het driver field. Men moet daarom goed opletten bij het schrijven van driver functies, dat het aantal en het type van de parameters van de verschillende functies, overeenkomen met de parameters die beschreven staan in de documentatie bij de GKS implementatie.
Door de interface tussen GKS en de drivers op deze manier uit te voeren, wordt het volgende bereikt: De wijze waarop GKS de driver functies aanroept, is onafhankelijk van het type werkstation, en eveneens onafhankelijk van de adressen van de driver functies in de driver module. Het enige dat GKS nodig heeft om de driver functies aan te roepen, is de pointer naar de, in die driver module gedeclareerde, structure van het type Screen. Deze pointer wordt de screen pointer genoemd. Verder moet GKS als parameter meegeven, de pointer naar de workstation statelist van het betreffende werkstation. GKS heeft de beschikking over de waarde van deze pointer aangezien GKS zelf de workstation statelist van dat werkstation gecreeerd heeft. Hoe krijgt GKS echter de beschikking over de waarde van de screen pointer?
Zoals in het vorige hoofdstuk reeds is behandeld, moet aan de open_ws functie (zie regel 11 in het testprogramma in hoofdstuk 7.4) als derde parameter, het adres van een functie uit de driver meegegeven worden. De open_ws functie zal dit adres gaan gebruiken om de bijbehorende functie aan te roepen. In het geval van het GEB001 werkstation zal de geb001() functie moeten worden aangeroepen. Aangezien deze functie in de GEB driver module gedefinieerd is, kan de functie beschikken over het adres van het driver field, dat in de driver module gedeclareerd is. De functie geb001() zal de GKS functie inst_scr (install screen pointer) aanroepen, met als parameters de type aanduiding van het geopende werkstation, en het adres van (of wel de pointer naar) het driver field van dit werkstation. De functie inst_scr zal er dan voor zorgen dat de betreffende screen pointer, die als parameter is doorgekregen, wordt aangebracht in de bijbehorende workstation description table. Dit is de workstation description table, waarvan het type als parameter bij de inst_scr functie aanroep is doorgegeven. Indien GKS dus een bepaalde driver functie wil adresseren, dan kan GKS de

benodigde screen pointer vinden in de, bij dat werkstation behorende, workstation description table.

8 Software voor de GKS Driver

8.1 Algemeen

Voor de omschrijving van de organisatie van de software op de Unix VAX van de THE, wordt verwezen naar hoofdstuk 6.1 Lokalisatie van de Files en hoofdstuk 6.2 Listings. Hierin wordt beschreven onder welke directories de software aanwezig is en hoe de, in bijlage A bijgevoegde, listings zijn ingedeeld.
Bij de ontwikkeling van de driver voor de grafische kaart, hierna genoemd de GEB driver, is de driver voor het AED512 grafische werkstation als uitgangspunt genomen. Dit werkstation bestaat namelijk uit een raster scan display met keyboard. Dit werkstation vertoonde de meeste overeenkomsten met het GEB grafisch werkstation. Bij de ontwikkeling van de GEB driver zijn alle files ten behoeve van de AED512 driver, vanuit de directory gks, gecopieerd naar de directory graf. Vervolgens zijn de namen van deze files gewijzigd, zodanig dat het voorvoegsel aed steeds gewijzigd werd in geb. Alleen die files van de AED512 driver, die niet zijn herschreven voor de GEB driver (dit zijn over het algemeen de files die input verzorgen vanuit het grafische werkstation), zijn ongewijzigd gebleven.
Aangezien in de oorspronkelijke implementatie van het C.W.I. vrijwel geen commentaar in de driver files aanwezig was, is extra commentaar bijgevoegd.
Het oorspronkelijke commentaar in alle files is echter ongewijzigd gelaten. Het oorspronkelijke commentaar staat in het Engels en is dus makkelijk te onderscheiden van het naderhand aangebrachte commentaar, dat in het Nederlands is gesteld.

8.2 Opbouw Driver Software

De files ten behoeve van de driver voor de grafische kaart staan in de directory /usr/users/elebst3/bert/graf. Alle driver files beginnen met het voorvoegsel geb. De files bestaan uit een negental header files en een c file, de file geb001.c. In de file geb001.c worden eerst een aantal gks header files geinclude, zoals de file cgksincl.h. Deze file zelf bestaat alleen maar uit include statements. Door deze file te includen worden een elftal GKS header files geinclude. Deze header files bevinden zich in de directory gks. Omdat zij worden geinclude in de file geb001.c moeten ze ook in de directory graf aanwezig zijn. Dit is verwezenlijkt door elk van de header files te linken naar de directory graf.
De file geb001.h is een file die als een van de eersten wordt geinclude in de file geb001.c. In deze file staan de definities van enkele belangrijke constanten, zoals de afmetingen van het scherm, die in het driver field gebruikt worden. Tevens staat hierin de definitie van een conditionele compilatie door middel van DEBUG. Vrijwel alle functies uit de driver zijn namelijk gedeclareerd als zijnde van het

type STATIC. Indien DEBUG gedefinieerd is, dan wordt STATIC gedefinieerd als zijnde niets. D.w.z. het wordt bij de functie definities gewoon weggelaten. Dit heeft als gevolg dat alle driver functies gewone external functies zijn, waaraan in andere files gerefereerd kan worden. Door middel van deze DEBUG faciliteit kunnen de functies in de driver dus extern geadresseerd en gedebugged worden. Als DEBUG echter niet gedefinieerd is, dan wordt STATIC, door middel van een define statement, gedefinieerd als zijnde static. Dit heeft als gevolg dat de driver functies van het type static worden, en dus onzichtbaar zijn buiten de geb001.c file. Dit is van belang als het programma foutvrij en klaar is. Op deze manier kunnen de functies uit de driver module nooit (ongewild) in conflict komen met gelijknamige functies uit bijvoorbeeld de GKS files, of met de functies van een applicatie programma, of met de functies van een andere driver die meegelinkt is. Omdat alle driver files header files zijn (met achtervoegsel .h), die geinclude worden in de geb001.c file, staan de driver functies dus feitelijk alle in dezelfde file, en kunnen dus wel aan elkaar refereren.
In de file geb001.c wordt de driver field structure van het type Screen, die in hoofdstuk 7.7 GKS/Driver Interface is behandeld, gedeclareerd en geinitialiseerd. De initialisatie van deze file is grotendeels overgenomen uit de file aed512.c uit de directory gks. De driver functies die in de GEB driver geimplementeerd zijn, dragen dezelfde namen als de oorspronkelijke functies uit de aed512 driver, waarbij de eerste drie letters aed vervangen zijn door het voorvoegsel geb.
In het driver field g_geb001 vinden we echter ook nog een aantal functies die het oude voorvoegsel aed dragen. Deze functies zijn in de GEB driver nog niet geimplementeerd. Het gaat hier voornamelijk om functies die de GKS input functies ondersteunen. Deze functies zijn nog niet geimplementeerd, aangezien zij nogal gecompliceerd zijn. Het is helaas duidelijk geworden uit mededelingen van mensen uit het Rekencentrum van de THE, dat er in de C-GKS implementatie van het C.W.I., en ook in de drivers, nogal wat fouten zitten. Het is daarom belangrijk te beginnen met het implementeren van een zo eenvoudig mogelijke versie van GKS op het THE-KUNix systeem.
Voorafgaande aan de declaratie van het driver field, worden de namen van de functies die in de initialisatie van het driver field voorkomen, gedefinieerd als zijnde adressen van functies die een boolean returnen. Hierdoor weet de compiler dat de identifiers in de driver field initialisatie, functie adressen zijn. Van de niet geimplementeerde driver functies, zijn de functienamen, door middel van #define statements, gedefinieerd als zijnde NULL. Indien GKS in een driver field als functie adressen de waarde NULL tegenkomt, dan ziet GKS hieraan dat deze driver functies niet zijn geimplementeerd. Het is belangrijk dat de namen van deze functies in de initialisatie niet vervallen, maar dat er in de plaats daarvan de NULL waarde voor wordt ingevuld. Indien deze functie adressen zouden worden weggelaten, dan zouden de

adressen van de wel geimplementeerde functies, in de initialisatie van het driver field op de verkeerde plaatsen komen te staan.
Ten slotte worden in de geb001.c file de 9 header files uit de directory graf geinclude. Deze files hebben allen het voorvoegsel geb.

8.3 Interface Driver/Basis-software Grafische Kaart

Aangezien de beschikbare driver functies bepaald worden door de Screen type definitie van het driver field, ligt de functie van de driver procedures vast. De functie van deze procedures evenals het aantal parameters, wordt namelijk bepaald door de interface tussen GKS en de driver. Bij de implementatie van de GEB driver, uitgaande van de AED512 driver, zijn de functie en de parameters van de aanwezige driver procedures dus ongewijzigd gebleven. De implementatie van de driver procedures is echter aangepast aan de op de vakgroep EB ontwikkelde grafische kaart. Deze wijzigingen waren in veel gevallen nogal ingrijpend, aangezien het AED512 werkstation moet worden aangestuurd als een Unix terminal, d.w.z. op basis van file IO naar een Unix file device. De GEB driver is echter herschreven voor implementatie in het systeem, zoals dat beschreven is in hoofdstuk 4 in figuur 4.7: Eenvoudige GKS implementatie onder Unix. Hierbij moet de driver dus rechtstreeks gelinkt worden met de LEVEL 0 basis software die de grafische kaart aanstuurt. Hierbij kwamen enige compatibiliteits problemen boven. Deze problemen hadden drie oorzaken:

- De C implementatie van het C.W.I. maakt veel gebruik van eigen type definities. Deze speciale type definities zijn echter niet gebruikt in de LEVEL 0 software.
- Het coordinaten systeem van de GKS werkstations heeft de oorsprong in de linker onderhoek van het beeldscherm. Het coordinaten systeem van de grafische kaart, d.w.z. het coordinaten systeem van de GDC's, heeft de oorsprong echter in de linker bovenhoek.
- Wanneer in een driver functie een punt met een x en een y coordinaat als parameter moet worden doorgegeven, wordt in het algemeen een pointer naar een structure, waarin de x en de y coordinaten zijn opgeslagen, doorgegeven. In de LEVEL 0 software worden in zo'n geval echter de x coordinaat en de y coordinaat beide als losse parameters doorgegeven.

De bovenstaande incompatibiliteits problemen zijn als volgt opgelost:
Alle driver functies moeten uiteindelijk eindigen in aanroepen van functies uit de LEVEL 0 software. Er is nu een soort interface laag gedefinieerd tussen de driver software en de LEVEL 0 software. Deze interface laag bestaat gedeeltelijk uit macro functies en tevens uit gewone functies. Alle interface functies zijn gedefinieerd in de speciaal daarvoor geconstrueerde file gebconv.h (geb conversion). De aanroep van deze conversie functies gebeurt geheel volgens de normen

van GKS, d.w.z. met als coordinaten stelsel de linker onderkant van het scherm, gebruik makend van de GKS type definities en gebruik makend van de GKS methode om een coordinaten punt als parameter door te geven. Binnnen de functie wordt dan de nodige omzetting gedaan, waarna de functies uit de LEVEL 0 software op de juiste wijze aangeroepen worden.

8.4 Initialisatie van de Grafische Kaart

De initialisatie van de grafische kaart vindt plaats door middel van de aanroep van de driver functie geberase. Als deze functie wordt aangeroepen wanneer het werkstation nog niet is geinitialiseerd, dan wordt hiermede het werkstation geinitialiseerd en het grafische beeldscherm schoon geveegd. Als deze functie wordt aangeroepen nadat de initialisatie al heeft plaats gevonden, zal alleen het grafische beeldscherm worden schoon geveegd. De initialisatie van de grafische kaart gebeurt op de volgende wijze:
Eerst wordt de functie init aangeroepen. Deze functie uit de LEVEL 0 software module, zorgt voor de gebruikelijke initialisering van de grafische kaart. Na de aanroep van de init functie, vindt echter nog een verdere initialisatie plaats. De functie fsllinestyle (function select linestyle) wordt aangeroepen. Hierdoor komt de grafische kaart in de linestyle mode te staan. Zoals uitgelegd in hoofdstuk 6.5 Werking Basis Software, kan de grafische kaart in 3 modes werken: De normal mode, de linestyle mode en de character mode. Iedere mode heeft zijn eigen variabelen. Multipixel (zie hoofdstuk 6.5) kan in elke mode aanstaan of uitstaan. Eveneens heeft elke mode de beschikking over zijn eigen modify tabellen die, in de terugkoppellus, bij het tekenen gebruikt worden. De normal mode werd gebruikt voor het tekenen van continu getrokken lijnen, de linestyle mode voor het tekenen van stippellijnen of wel onderbroken lijnen, de character mode voor het tekenen van characters. Dit concept waarbij iedere mode zijn eigen variabelen of wel attributen bezit, is bij de huidige implementatie van de driver verlaten.
In de huidige driver implementatie werkt de grafische kaart steeds in de linestyle mode. Ook continue ononderbroken lijnen worden in de linestyle mode getekend. Hiervoor moet het linestyle register, dat het zich herhalend 16 bits patroon van de lijn bevat, geladen worden met allemaal enen. Indien er characters moeten worden getekend, wordt de grafische kaart kortstondig gedeeltelijk in de character mode geschakeld. De attributen van de character mode worden echter niet gebruikt. Onder alle omstandigheden worden namelijk de attributen van de linestyle mode gebruikt. De kleur waarin wordt getekend kan centraal worden ingesteld, maar zal dus bij het tekenen van lijnen en characters dezelfde kleur zijn. De multipixel mode zal altijd aan staan. Deze vereenvoudigingen hebben de volgende redenen:
- Gezien de in hoofdstuk 8.2 reeds genoemde fouten die in de GKS implementatie aanwezig zijn, is het belangrijk om bij het uittesten van het systeem met een zo eenvoudig moge-

lijke driver te beginnen. Hiermede wordt voorkomen dat onnodig extra fouten worden geintroduceerd.

- De huidige DI/DD interface tussen GKS en de drivers, zoals deze door het C.W.I. is geimplementeerd, maakt geen gebruik van attributen die intern in een grafisch werkstation ingesteld zouden kunnen worden. De driver functies setlatt, setmatt, settatt, setfatt (respectievelijk: set line, marker, text en fill area attributes) wekken wel deze indruk. Men zou verwachten, dat hiermee de kleur, de dikte en het type van de te tekenen lijnen zou kunnen worden ingesteld (setlatt), en onafhankelijk hiervan de attributen (kleur, type etc.) van de markers (setmatt), de tekst (settatt) etc..
  In werkelijkheid wordt door deze functies alleen de, op het werkstation gebruikte, kleur centraal ingesteld. D.w.z. deze kleur wordt door alle tekenfuncties, zoals lijnen, markers en tekst, gebruikt. De centrale kleur wordt door de setlatt etc. functies geplaatst in de variabele s_ccol in het driver field, en geoutput naar het werkstation. In een aantal gevallen wordt ook het lijntype ingesteld, echter ook hier weer centraal, dus voor alle tekenfuncties gelijk (zie voor deze functies de file aedatt.h).
  Bij het tekenen van lijnen en markers, worden aan de drawline en drawmark functie als parameters o.a. meegegeven het lijn / marker type en de lijn breedte, respectievelijk de marker grootte. Deze parameters kunnen, afhankelijk van de variabele gk_asf uit de GKS statelist (zie hoofdstuk 7.5), afkomstig zijn uit de GKS statelist (unbundled attributen) of uit de workstation statelist (bundled attributen). Hierbij moet nog worden opgemerkt, dat het instellen van het lijntype m.b.v. de setlatt functie, zoals in de C.W.I. implementatie gebeurt, geen zin heeft. Het lijn type wordt namelijk bij iedere aanroep van de drawline functie, reeds als parameter meegegeven.

De bovengenoemde vereenvoudiging, waarbij de grafische kaart steeds in dezelfde mode staat, is als volgt verwezenlijkt:
Bij het tekenen wordt steeds een en dezelfde modify tabel geselecteerd. Dit is modify tabel nummer 4, die normaal alleen in de linestyle mode wordt gebruikt. Wanneer er characters op het beeldscherm worden afgebeeld, zal deze zelfde tabel ook geselecteerd blijven. De multipixel variabele is, zoals reeds gezegd, steeds aangeschakeld, zodat continu plaats nr 15 van de vierde modify tabel geselecteerd zal zijn. De kleur index die op deze plaats in de tabel staat, is dus de kleur index die gebruikt zal worden bij het tekenen, onafhankelijk of er characters of wel lijnen op het beeldscherm worden afgebeeld. Het zal duidelijk zijn dat op deze manier grote delen van de hardware van de grafische kaart niet worden gebruikt. Indien echter de huidige driver implementatie is uitgetest en werkt, kunnen later de toevoegingen worden aangebracht, om de verdere mogelijkheden van de hardware op de grafische kaart te gebruiken.

9 Ontwikkelomgeving

9.1 Globaal Overzicht

Editen en Compileren in Nijmegen

Bij het tot stand komen van de ontwikkelde software, heeft de ontwikkelomgeving een belangrijke rol gespeeld. Onder de ontwikkelomgeving wordt verstaan de hardware en software hulpmiddelen, die bij de software ontwikkeling gebruikt worden.
Bij de aanvang van het afstudeerwerk, was er nog geen Unix systeem op de TH Eindhoven aanwezig. Alle software werd toen ontwikkeld op een Unix systeem van de Katholieke Universiteit Nijmegen. De in Nijmegen ontwikkelde software staat in de directory bert. De directory bert is een subdirectory van de inlog directory ebthe.
Op de vakgroep Digitale Systemen van de TH Eindhoven is gebruik gemaakt van een Altos computer systeem. Het Altos computer systeem was aangesloten op een modem. Met behulp van het modem werd er via een telefoonlijn een verbinding tot stand gebracht, tussen het Altos computer systeem en de Unix computer in Nijmegen. Alle communicatie met de Unix computer in Nijmegen verliep via het kermit file transmission programma, dat op de Altos computer was opgestart. Hiertoe werd het kermit programma door middel van het connect commando in de transparante mode gezet.
Met behulp van de em line editor werden de C programma's in Nijmegen aangemaakt. Deze em editor is een in Nijmegen ontwikkelde versie van de standaard Unix ed editor, waaraan enkele uitbreidingen zijn toegevoegd. Vervolgens werden de programma's op het Unix systeem in Nijmegen gecompileerd. Het gecompileerde programma moest uiteindelijk op het M68000 systeem in Eindhoven worden uitgetest. Dit systeem is reeds in hoofdstuk 4 beschreven. Dit systeem accepteert echter alleen binaire executable programma modules in een speciaal formaat. Het zogenaamde S-record formaat. Op het Unix systeem in Nijmegen is een programma aanwezig, dat zorg draagt voor de conversie van een binaire executable object module, zoals die door de C compiler wordt geproduceerd, naar het S-record formaat. Dit programma heet xmacs68. Overigens kon op het Unix systeem in Nijmegen niet de standaard C compiler worden gebruikt. Deze compiler produceert namelijk code die alleen draait op een PDP-11 computer. In plaats van de standaard C compiler cc is dan ook gebruik gemaakt van de cc68 compiler, die code produceert geschikt voor de M68000 microprocessor. Nadat de module tot het correcte formaat geconverteerd was, werd de module met behulp van kermit overstuurd naar het Altos computer systeem in Eindhoven. Hiertoe werd het kermit programma op het Unix systeem in Nijmegen opgestart in de send mode, en het kermit programma op de Altos computer in Eindhoven in de receive mode. De module kwam dan uiteindelijk terecht op een 8 inch diskette op het Altos systeem.

Editen in Eindhoven en Compileren in Nijmegen

Omstreeks het tweede kwartaal van 1985 werd op de TH Eindho-

ven in het Rekencentrum de Ultrix VAX in gebruik genomen. Dit Unix systeem heeft de beschikking over de vi editor. De vi editor is een screen oriented editor. D.w.z., dat bij het editen van een tekst de hele tekst op het beeldscherm verschijnt. Zodoende kan men met de cursor over het hele scherm bewegen en op elke willekeurige plaats wijzigingen aanbrengen. In Nijmegen was er geen screen editor beschikbaar. De ed en de em editors, die in Nijmegen beschikbaar waren, zijn beide line oriented editors. D.w.z. dat men met deze editors slechts een regel tegelijk kan editen. Uiteraard is het editen met behulp van een line editor een zeer tijdrovende bezigheid. Zodra er dus in het Rekencentrum van de TH Eindhoven een Unix systeem beschikbaar kwam, is direct begonnen met het overbrengen van de files uit Nijmegen naar Eindhoven. Het bleek echter al vrij snel dat de speciale compileer programma's cc68 en de loader ld68, die object code produceren voor de M68000 processor, niet aanwezig waren op de Unix VAX in Eindhoven. Er is toen getracht deze programma's vanuit Nijmegen over te brengen naar Eindhoven. Dit is echter vanwege compatibiliteits problemen niet gelukt. Er was binnen het kader van mijn afstudeerwerk geen tijd meer beschikbaar om in deze problematiek te duiken. Momenteel probeert een stagiaire de compileerprogramma's voor de M68000 microprocessor over te brengen vanuit Nijmegen naar Eindhoven.
Er deed zich dus de volgende situatie voor: De files moesten in Eindhoven geedit worden. Iedere keer na het aanbrengen van wijzigingen en toevoegingen aan het programma, moesten de files weer overgestuurd worden naar Nijmegen om daar gecompileerd te worden. Na compilatie en conversie naar het formaat van S-record, moest de module weer naar de Altos computer in Eindhoven worden overgezonden. Het zal duidelijk zijn dat deze cyclus om tot een nieuw programma module te komen, een vrij tijdrovende aangelegenheid was. Dit hele proces is dan ook zo veel mogelijk geautomatiseerd.
De Unix makefile biedt de mogelijkheid om automatisch bij te houden welke source files geedit worden. Via een speciaal make commando konden de gewijzigde files automatisch worden overgestuurd naar Nijmegen. In Nijmegen werden ook functies in de makefile aangebracht, zodat het mogelijk werd door middel van het "make gdc" commando automatisch te compileren. Door middel van het send commando werd de gecompileerde module vervolgens automatisch geconverteerd en terug gestuurd naar Eindhoven. Ten gevolge van de enorme tijdwinst bij het editen, dat nu in Eindhoven kon plaats vinden, en waarvoor dus ook geen telefoonverbinding meer nodig was, heeft deze benadering toch een netto tijdwinst opgeleverd. Bovendien is het compileren van de modules in Nijmegen per definitie een tijdelijke zaak. In de toekomst moet het programma namelijk geimplementeerd worden op het Unix systeem van de vakgroep EB. Compilaties in Nijmegen zullen in de toekomst dan ook niet meer nodig zijn. De functies in de makefile waarmee onder andere het gehele compilatie en edit proces geautomatiseerd is, worden in hoofdstuk 9.2 nader behandeld. Eerst zullen we echter nader ingaan op de uiteindelijke software modules voor het M68000 systeem.

Software Modules voor het M68000 Systeem

De naam van de gecompileerde module was steeds gdc.sr. Gdc slaat op grafical display controller. De module was namelijk bestemd om de GDC's op de grafische kaart aan te sturen. Het achtervoegsel sr slaat op S-record. De file gdc.sr werd meteen na ontvangst gerenamed volgens het volgende principe. De nieuwe naam van de file werd gdc met daarachter de datum waarop de file was gecompileerd. De datum bestaat uit een viercijferig getal, met als eerste twee cijfers de maand aanduiding en als laatste twee cijfers de dag aanduiding. Indien een bepaalde versie van het programma bijvoorbeeld op 22 maart gecompileerd was, dan werd de naam van de gdc.sr file gerenamed tot gdc0322. Deze handelwijze was bij de software ontwikkeling zeer essentieel. Dit zal hieronder nader worden toegelicht.

De grafische kaart was een prototype dat geheel opgebouwd was met behulp van de wire wrap techniek. Door deze techniek konden eenvoudig storingen optreden, bijvoorbeeld doordat de isolatie van een draadje was versleten, ofwel doordat er twee wire wrap pennen van een IC voetje kortsluiting maakten. Ook zijn er tijdens de ontwikkeling storingen opgetreden, die te wijten waren aan timing fouten op de print, doordat sommige timing processen te kritisch waren ingesteld.

Tijdens de software ontwikkeling werden er uiteraard steeds wijzigingen en toevoegingen in de software aangebracht. Indien nu een nieuwe versie van de software niet meer correct functioneerde, dan kon dit twee oorzaken hebben: Ofwel er was een fout gemaakt in het software programma, ofwel er was een storing opgetreden in de grafische kaart.

Voordat er uitbreidingen en toevoegingen aan de software werden aangebracht, werd eerst de huidige versie van de software uitgetest. Van iedere uitgeteste versie van het programma werden de listings van de sources en de 8 inch diskette met de gecompileerde modules bewaard. De gecompileerde modules zijn herkenbaar aan de datum aanduiding die in de naam is verwerkt.

Indien er nu een fout optrad in het grafische systeem, dan werd de nieuwe software module vervangen door een oude software module, waarin geen wijzigingen aangebracht waren, en dat volledig was uitgetest. Indien de fout met het oude programma mog steeds aanwezig was, dan was het hiermede direct duidelijk, dat de fout lag aan de grafische kaart en niet aan de software. Deze situatie heeft zich in de praktijk enkele malen voorgedaan. Ook is het enkele malen gebeurt, dat de frekwentie generator, die aangesloten was op de grafische kaart, versteld was. Deze frequentie generator had zeer vele mogelijkheden en veel ingewikkelde instelknoppen, die af en toe per ongeluk of expres door anderen werden versteld. Het verdient dan ook aanbeveling om bij een volgende versie van de grafische kaart de pulsgenerator direct op de kaart te bouwen.

Bij het beeindigen van het afstudeerwerk is aan mijn coach P.H.A. van de Putten de bovengenoemde 8 inch diskette overgedragen, samen met een kopie van deze diskette. Tevens is

overgedragen een 5 inch IBM PC diskette met een back-up van alle ontwikkelde software op de Unix VAX in het Rekencentrum van de T.H.E.. Hierbij zijn voorts nog twee extra kopieen bijgevoegd.

9.2 Makefile

De Unix makefile is in het vorige hoofdstuk reeds ter sprake gekomen. Het Unix commando make maakt gebruik van de gegevens, die in de makefile zijn opgeslagen. Het is niet de bedoeling om hier in detail uit te leggen hoe het make commando werkt, en wat voor mogelijkheden er zijn met de makefile. Dit alles is reeds uitvoerig beschreven in de Unix manuals lit. 5) en 6). Bovendien is de makefile op een zodanige wijze van commentaar voorzien, dat de functie van alle make commando's die met de huidige makefile mogelijk zijn, voor zich spreken. De listing van de makefile is in bijlage A bijgevoegd. De makefile is gelocaliseerd op het Unix systeem van de TH Eindhoven (Unix VAX in Rekencentrum) in de directory /usr/users/elebst3/bert/graf. De makefile op het Unix systeem in Nijmegen is een subset van de makefile in Eindhoven.
We zullen hieronder de werking van de belangrijkste make commando's behandelen. In het algemeen kunnen we stellen, dat er zo veel mogelijk gebruik is gemaakt van de mogelijkheden die het make commando biedt, om de programma compilatie te automatiseren. Als een software module bestaat uit verschillende files, die tezamen gecompileerd en geladen moeten worden, zal op de volgende wijze van de make functies gebruik gemaakt worden. De compilatie wordt gestart door middel van een make commando. De make functie zal nu onderzoeken of er files zijn, die na de meest recente compilatie gewijzigd zijn. De make functie zal dan automatisch deze gewijzigde files, en alle files die van de gewijzigde files afhangen, compileren. De wijze waarop files van elkaar afhangen kan in de makefile gespecificeerd worden.
We zullen nu de belangrijkste make commando's uit de besproken makefile gaan behandelen. Bij de verklaring van de make commando's zal regelmatig gerefereerd worden aan de software modules die in hoofdstuk 4 in de verschillende figuren zijn gebruikt.
De belangrijkste make commando's zijn de volgende:

make gdc
: Dit commando kan alleen op het Unix systeem in Nijmegen gebruikt worden, aangezien dit commando gebruik maakt van de C compiler die de object code voor de M68000 processor levert. Het commando draagt zorg voor automatische compilatie van de LEVEL 0 software uit figuur 4.4. De resulterende object module heet gdc. Aan de C compiler wordt de volgende variabele meegegeven: -DSYSTEM=M68000. -D is een define statement. Hiermede wordt de variabele SYSTEM gedefinieerd als zijnde M68000. De waarde van de variabele SYSTEM wordt gebruikt bij de conditionele compilatie. In de source

files wordt op de volgende manier gebruik gemaakt van conditionele compilatie:

```
#if (SYSTEM==M68000)
.....
.....
.....
#endif
```

of ook:

```
#if (SYSTEM==UNIX)
.....
.....
.....
#endif
```

Dit mechanisme van conditionele compilatie werkt als volgt:
In het eerste geval wordt in het #if statement gecontroleerd of SYSTEM de waarde M68000 heeft. Zo ja, dan worden alle statements voorafgaande aan het eerst volgende #endif gecompileerd. Zo nee, dan worden deze statements niet gecompileerd. Op analoge wijze kan worden getest of SYSTEM de waarde UNIX heeft.
Door aan de compiler door middel van -DSYSTEM=...., SYSTEM de waarde M68000 of UNIX te geven, kan met behulp van de conditionele compilatie statements automatisch de juiste software module gecreeerd worden. D.w.z. de module die geschikt is voor het M68000 systeem ofwel geschikt voor Unix.

make gdcunix
Met dit commando wordt de LEVEL 0 software module uit figuur 4.6 gecreeerd. De resulterende module heeft de naam gdcunix. Hierbij wordt aan de C compiler de parameter -DSYSTEM=UNIX meegegeven.

make gdcgks
Met dit commando wordt de LEVEL 0 software module uit figuur 4.7 gecreeerd. De module krijgt de naam gdcgks. De parameter voor de C compiler is hier eveneens -DSYSTEM=UNIX.

make gebdriver
Met dit commando wordt de GKS driver gecreeerd die de grafische kaart kan aansturen. De driver bevindt zich in de file geb001.c. Als eerste wordt deze file gecompileerd, waarbij de gecompileerde file de naam geb001.o krijgt. Deze file wordt dan met behulp van het Unix ld commando geladen met de output module van het make gdcgks commando. Hierdoor worden dus de modules geb001.o en de module gdcgks samen geladen. De resulterende output module krijgt de naam gebdriver. Deze module kunnen we in figuur 4.7 terugvinden als de combinatie van de module DRIVER en de module LEVEL 0.

De Unix ld loader is aangeroepen met de -r flag, zodat hij relocatable object code produceert. Dit is nodig aangezien de gebdriver module tezamen met GKS en de GKS APPLICATIE module moet worden geladen tot een executable object file (zie fig. 4.7).

Verder zijn er nog een aantal make commando's die gebruikt kunnen worden om de verschillende sources te printen. We zullen hieronder een selectie laten volgen van de belangrijkste print commando's.

make prall
Print alle sources uit de directory graf. D.w.z. alle sources voor de LEVEL 0 software, de OPERATOR IO software en de DRIVER software.

make prallgraf
Print alle sources uit de LEVEL 0 en de OPERATOR IO software.

make pralldriv
Print alle sources van de DRIVER software.

make print
Print alleen die files, die sinds de laatste printout gewijzigd zijn.

De C compiler voert geen volledige syntax check uit. Zo zal de C compiler bij een functie aanroep niet controleren, of het aantal parameters en het type van de parameters correct is, indien deze functie in een andere file is gedefinieerd. Met behulp van de Unix utility lint worden dit soort dingen wel gecontroleerd. Lint voert een zeer nauwkeurige syntax check uit. Het is dan ook zeer belangrijk om alle gecompileerde modules uiteindelijk een keer door lint te halen. Hiervoor zijn de volgende make commando's gerealiseerd:

make lint
Hiermee worden alle sources ten behoeve van de LEVEL 0 software voor het M68000 systeem op syntax gecontroleerd. Aan lint wordt, analoog als bij de compilatie, door middel van de -D flag doorgegeven dat SYSTEM de waarde M68000 krijgt. Op deze manier zal ook lint bij de syntax check gebruik maken van de conditionele compilatie.

make lintunix
Syntax check als hierboven, echter voor de LEVEL 0 software onder Unix.

make lintdriva11
Lint syntax check van de LEVEL 0 software en de DRIVER software uit figuur 4.7, voor applicatie onder Unix met GKS. Aangezien de driver functies functie aanroepen doen van de functies die in de LEVEL 0 software gedefinieerd zijn, moeten bij een lint syntax check alle

files uit de LEVEL 0 en de DRIVER software modules tegelijkertijd door lint gecontroleerd worden. Alleen op deze wijze kan een controle plaats vinden van het aantal en het type van de parameters waarmee de verschillende functies uit de verschillende modules worden aangeroepen.

Ten slotte zijn er een aantal make commando's gerealiseerd, die gebruik maken van kermit, om files te transporteren vanaf een Unix systeem naar een Altos computer, ofwel naar een IBM PC voor back-up doeleinden.

make backupnij

Met dit commando wordt het compilatie proces van de M68000 software ondersteund. De sources moeten namelijk in Eindhoven worden geedit en vervolgens worden doorgestuurd naar Nijmegen om gecompileerd te worden. Met dit commando worden automatisch die files, ten behoeve van de M68000 software, overgezonden naar Nijmegen, die sinds de laatste keer dat het make backupnij commando gegeven werd gewijzigd zijn. Hiertoe wordt kermit onder Unix in Eindhoven opgestart en stuurt automatisch de juiste files over naar de ontvangende kermit. De ontvangende kermit zal in het algemeen een kermit programma zijn dat op de Altos computer op de vakgroep EB draait. Alleen via deze Altos computer, kan via een modem een telefonische verbinding met het Unix systeem in Nijmegen worden verkregen. De datum van verzending wordt meegestuurd in de file date. De files die op deze wijze op de Altos ontvangen zijn, kunnen op de volgende manier naar Nijmegen doorgestuurd worden. Hiertoe moet op de Altos kermit worden opgestart en in Nijmegen worden ingelogd met ebthe. Op het Unix systeem in Nijmegen wordt dan eerst het cd bert commando, en vervolgens het make install commando gegeven. Hierdoor krijgt de gebruiker automatisch op zijn scherm aangegeven, welke toetsen hij moet indrukken om kermit over te schakelen naar de send mode om de files naar Nijmegen over te zenden. Het make install commando zal dan automatisch zorg dragen voor de ontvangst van de files.

make backupall

Aangezien elk computer systeem storingen kan vertonen, is het belangrijk regelmatig back-up's te maken. Dit te meer gezien het feit, dat het Rekencentrum te kennen heeft gegeven, niet van plan te zijn regelmatig zelf tape back-up's te maken. Door middel van het make backupall commando, wordt automatisch, met behulp van kermit, een back-up gemaakt van alle files voor de grafische software. De back-up kan momenteel gemaakt worden op 8 inch diskettes op het Altos systeem. Een tweede mogelijkheid is het maken van back-up's op 5 inch diskettes met behulp van een IBM PC. Hiertoe moet op het bewuste systeem kermit worden opgestart en door middel van het connect commando kermit in de transparante mode gezet worden. Vervolgens moet worden inge-

logd op het THE net en moet een verbinding tot stand gebracht worden met de Unix VAX in het Rekencentrum. Onder Unix moet dan het make backupall commando gegeven worden. Dit commando zorgt er voor, dat automatisch op het scherm van de gebruiker de boodschap komt, dat de files verzonden zullen worden, en welke commando's de gebruiker aan zijn eigen kermit moet geven om kermit in de receive mode te zetten. Hierna zullen automatisch alle grafische files vanaf het Unix systeem overgestuurd worden naar de ontvangende kermit op de IBM PC, ofwel op de Altos.

make backup

Ook dit commando is bedoeld voor het maken van back-up's. Dit commando wordt gebruikt als men reeds een keer een back-up heeft gemaakt van alle files met behulp van het backupall commando. In het vervolg kan men dan volstaan met het make backup commando. Het make backup commando werkt analoog als het make backupall commando, waarbij echter alleen die files worden verstuurd voor back-up, die sinds het laatste backup of backupall commando gewijzigd zijn.

9.3 Utilities

Als hulpmiddel bij de ontwikkeling van de software zijn er een aantal utilities ontwikkeld. Deze utilities staan op de Ultrix VAX van het Rekencentrum van de TH Eindhoven onder de inlognaam elebst3. Onder deze inlog directory staan de utilities in de directory bert/bin. In de bin directory is tevens een makefile aangebracht. In deze makefile staan functies om de utilities uit de bin directory uit te printen en om er back-up's van te maken, analoog als de wijze waarop dit in het vorige hoofdstuk is behandeld. Iedere utility is van commentaar voorzien, zodat duidelijk is wat het doel is van de utility en hoe de utility werkt. De listings van deze utilities zijn in de bijlage bijgevoegd. In hoofdstuk 6.2 Listings is beschreven hoe de listings georganiseerd zijn. De vier belangrijkste utilities zullen hieronder besproken worden.

prprocs

Bij de bestudering van de GKS software kwam een probleem al spoedig boven. De GKS listings bestaan uit een tweehonderdtal files, welke uitgelist een enorme stapel software opleveren. Het aantal gedefinieerde procedures en macro functies in de GKS software bedraagt ongeveer zeshonderd stuks. Bij de bestudering van een willekeurig stuk software uit de GKS listings, bleek al spoedig dat er continu procedures en macro functies werden aangeroepen, waarvan de functie onduidelijk was. Evenmin kon worden achterhaald in welke files zij gedefinieerd waren. De behoefte naar een overzicht van alle gedefinieerde procedures, met bij alle procedures informatie over de plaats waar zij gedefinieerd zijn,

ontstond al spoedig. Deze informatie werd bij de GKS implementatie niet bijgeleverd. Ook onder Unix zijn dergelijke utilities niet beschikbaar. Het enige wat Unix ondersteunt is het maken van cross reference lijsten. Deze geven van elk C type variabele dat gebruikt wordt, alle plaatsen weer waar zij worden aangeroepen. Het betreft dus niet alleen de functies maar ook alle andere variabelen. Bovendien wordt elke plaats waar zij gebruikt worden gelist. Het zal duidelijk zijn dat de, op deze manier geproduceerde, cross reference lijsten volkomen onbruikbaar zijn, omdat zij voor 90% overbodige informatie geven. Er is daarom een hulp utility ontwikkeld om de definitie plaatsen van de procedures te listen.
Deze utility is genaamd prprocs, wat staat voor print procedures. De utility wordt aangeroepen met prprocs, gevolgd door de filenamen die op procedure definities onderzocht moeten worden. Een handig commando is hierbij prprocs *, waarmee alle in de huidige directory voorkomende files op procedures definities worden onderzocht. Alle gevonden procedures worden vervolgens op alfabetische volgorde geoutput naar de Unix standard output. Iedere procedure wordt gevolgd door de naam van de file waarin hij gedefinieerd is en het regelnummer in de file. Op deze wijze zijn alle GKS procedures gelist, welke listing de bovengenoemde zeshonderd procedures opleverde. Ook van de tijdens het afstudeerwerk ontwikkelde software, zijn alle gedefinieerde procedures m.b.v. de prprocs utility gelist. Dit als extra hulpmiddel bij de bestudering en uitbreiding van de software door volgende afstudeerders.

prdefs

Een volgend probleem bij de bestudering van de GKS software, was het voorkomen van variabelen waarvan de functie onduidelijk was. Vaak bleek achteraf, dat dit variabelen waren, die op een andere plaats door middel van een #define statement gedefinieerd waren. Er is daarom een tweede utility ontwikkeld genaamd prdefs. Deze utility wordt aangeroepen door middel van prdefs, gevolgd door de namen van de files die op het voorkomen van definitie statements onderzocht moeten worden. Handig is hierbij het gebruik van prdefs *, waarmee alle files uit de huidige directory op definitie statements worden onderzocht. Alle gevonden definitie statements worden geoutput, op alfabetische volgorde van de variabele die gedefinieerd is. Het hele definitie statement wordt geoutput, voorafgegaan door de naam van de file waarin de definitie statement is gevonden en het regelnummer in de file. Dit leverde bij de GKS software ca negenhonderd define statements op.

prman

Met behulp van de prman utility (print manual), kan de on line Unix manual uitgeprint worden. De aanroep van prman is analoog aan het Unix man commando, echter met

weglating van de naam van een commando waarvan man de manual zou moeten listen. Deze shell procedure voert het Unix man commando namelijk uit, voor alle in het systeem aanwezige Unix commando's. Met behulp van het man commando kan men van een willekeurig Unix commando de werking opvragen. Het betreffende onderdeel uit de Unix manual wordt dan door het man commando op de standard output geplaatst. De parameters die aan het prman commando kunnen worden meegegeven, zijn exact dezelfde als die aan het man commando kunnen worden meegegeven. B.v. "prman 2" print de Unix manual uit voor alle in het systeem voorkomende system calls.

outall

Met behulp van het outall commando wordt automatisch de listing geregenereerd van alle, tijdens mijn afstudeerperiode verwezenlijkte, software. Het outall commando is gebruikt bij het verwezenlijken van de listings uit de bijlage.

Aangezien een aantal van de door mij ontwikkelde utilities nuttig kunnen zijn voor algemeen gebruik, zijn de belangrijkste utilities in de eleb directory ondergebracht. Deze directory is de centrale directory van de vakgroep EB. Het volledige padnaam van deze directory is /usr/users/eleb. In deze directory kan alleen worden geschreven door de Unix system manager van de vakgroep EB. De leden van de vakgroep EB kunnen deze directory echter wel lezen. Degenen die deze directory kunnen lezen, zijn de leden van de Unix groep eleb. Dit zijn al diegenen die inloggen onder de naam elebst... en de naam elebmw... . Hierbij staat ... voor een getal, bijvoorbeeld elebmw3. St is de afkorting voor student en mw is de afkorting voor medewerker. De genoemde utilies zijn in de directory eleb ondergebracht in de subdirectory bin. In deze directory is ook het file transport programma kermit ondergebracht.
Indien men nu gebruik wil maken van de utilities uit de directory eleb/bin, dan moet men in de inlog directory in de file .profile, aan het padnaam van PATH de file /usr/users/eleb/bin toevoegen. Hoe een en ander precies geimplementeerd moet worden, kan men zien in de .profile file uit de directory eleb. Het file transport programma kermit is ook in de directory bin opgenomen, aangezien kermit niet bij de standaard Unix system utilities behoort. Er kan dus alleen gebruik gemaakt worden van kermit door PATH zodanig in te stellen, dat ook de utilities uit de eleb/bin directory kunnen worden gebruikt. In de directory eleb/bin is ook de file README toegevoegd. Deze file bevat alleen maar commentaar. In deze file is voor elke utility die in de bin directory voorkomt, uitleg gegeven over de functie van de utility en de syntax waarmee de utility aangeroepen moet worden. Alle files uit deze bin directory inclusief de README file zijn in een aparte listing bij het verslag bijgevoegd.

10 Follow-up

De werkzaamheden aan het grafische systeem zullen moeten worden voortgezet door een volgende afstudeerder. Voor deze afstudeerder zijn er de volgende werkzaamheden aan het grafische systeem te verrichten:

- In het grafische systeem zoals dat momenteel operationeel is in het M68000 systeem (zie hoofdstuk 4 figuur 4.4), zitten nog enkele fouten:
  - In de grafische kaart is een structurele hardware technische fout aanwezig. Het pixel panning mechanisme is technisch niet correct uitgevoerd. Het mechanisme van pixel panning is reeds verklaard in hoofdstuk 6.4 Commando Interpreter. De pixel panning is namelijk gerealiseerd door de horizontale beeldsynchronisatie te verschuiven. Voor de technische details wordt verwezen naar lit. 1) afstudeerverslag M.L. Verhoef. De schakeling werkt goed indien er geen gebruik wordt gemaakt van het zoomen (uitvergroten van het beeld). Indien het beeld echter wordt uitvergroot b.v. met een faktor 2, dan zou het pixel panning mechanisme ook met twee pixels tegelijkertijd moeten gaan pannen. Bij de huidige schakeling blijft de pixel panning echter met een pixel tegelijkertijd gaan.
  - Als de grafische monitor in de interlaced mode is ingesteld, flikkert het beeld. Door middel van het d commando (display) kunnen de parameters van de grafische monitor interaktief worden ingesteld. Wanneer de noninterlaced mode wordt ingeschakeld, dan houdt het flikkeren van het beeld op. Nu worden echter de verhoudingen van het beeld verstoord. In verticale richting wordt het beeld nu namelijk een faktor 2 uitvergroot. Het is niet duidelijk wat de oorzaak van deze problematiek is. Dit zal nader moeten worden onderzocht.
  - De GDC biedt de mogelijkheid om twee windows tegelijkertijd op het scherm af te beelden. Door de software wordt deze feature ondersteund. Het is echter gebleken dat het tweede window alleen in de noninterlaced mode op het scherm te krijgen is. Indien in de interlaced mode met twee windows wordt gewerkt, dan is er geen beeld op het scherm te krijgen.
  - Door middel van de init() procedure wordt de grafische kaart geinitialiseerd. Hierbij doet zich soms het verschijnsel voor, dat de kleurindex 0 in de colour look-up table op de waarde 15 wordt geinitialiseerd (d.w.z. dat alle bits op 1 geset zijn) in plaats van op 0. Er zit geen enkele systematiek in het optreden van deze fout. Waarschijnlijk is de oorzaak hiervan timings problemen. In de grafische kaart zijn in de loop der tijd namelijk nog verschillende andere problemen opgetreden, welke inmiddels grotendeels door de stagiaire Ernst Loskens zijn opgelost. Ook hier was in het algemeen de oorzaak

timingsproblemen en het niet goed verwerkt worden van de hoogfrekwente klok pulsen.
Het is ook niet uitgesloten dat de GDC's bugs vertonen. Een bug is in ieder geval vastgesteld. Dit betreft het niet betrouwbaar zijn van de status bits uit het status register van de teken GDC. Het Draw bit, dat aangeeft wanneer de GDC bezig is met tekenen, is namelijk niet betrouwbaar. In de software wordt elke test op dit bit dan ook uitgevoerd, totdat er vier keer dezelfde waarde is ontvangen. De toegepaste GDC's zijn dan ook al vrij oud.

- De technische literatuur van de GDC's moet vernieuwd worden. Deze literatuur is nu namelijk reeds vrij oud en zeker niet meer up to date. Zie lit. 7) NEC Manual.

- De stagiaire Ernst Loskens is bezig met het toevoegen van verbeteringen aan de grafische kaart. Dit werk zal waarschijnlijk voortgezet moeten worden. Het is in ieder geval belangrijk, dat er op de grafische kaart de mogelijkheid wordt toegevoegd om het grafische geheugen uit te lezen. Hiervan wordt namelijk door GKS via de driver functies gebruik gemaakt.

- Een noodzakelijke toevoeging is ook het onderbrengen van de hoog frequente klok generator op de grafische kaart. Tot nu toe moest namelijk met een aparte frequentie generator worden gewerkt, wat bijzonder onhandig is.

- Alle uitbreidingen en verbeteringen, die aan de grafische kaart worden aangebracht, zullen softwarematig door nieuw te ontwikkelen functies moeten worden ondersteund.

- De OPERATOR IO software laag en de LEVEL 0 software laag zullen, in het door de vakgroep EB nieuw ontwikkelde Unix systeem, moeten worden uitgetest. Zie voor deze software hoofdstuk 4 figuur 4.6.

- Alle benodigde software voor het implementeren van GKS op het Unix systeem van de vakgroep EB is ontwikkeld. Deze software is weergegeven in hoofdstuk 4 figuur 4.7: Eenvoudige GKS implementatie onder Unix. Deze software zal uitgetest moeten worden, waarbij eerst GKS op het Unix systeem dient te worden geinstalleerd. Zie hiervoor lit. 4 Installation Guide for C-GKS.

- Bij de, voor het GEB grafische werkstation, ontwikkelde driver, zullen de input functies nog moeten worden toegevoegd. Hierbij kunnen de overeenkomstige input functies uit de AED512 driver als uitgangspunt worden genomen. De AED512 driver is beschikbaar in de huidige GKS implementatie.

- Uiteindelijk zal het grafische werkstation volgens figuur 4.9 uit hoofdstuk 4 moeten worden geimplementeerd. Hiervoor zal zowel nieuwe hardware als software moeten worden

ontwikkeld. Voor de beschrijving van dit systeem wordt verwezen naar hoofdstuk 4.5 Toekomstig Systeem.

11 Conclusies

Het afstudeerwerk heeft geleid tot een afgerond pakket software. De software is operationeel in het M68000 systeem. Ook voor implementatie onder Unix zijn twee afgeronde software modules beschikbaar gekomen. Een voor gebruik in samenwerking met de grafische kaart als stand alone unit, en een voor de implementatie van GKS.
Het Unix systeem is pas sinds kort operationeel, zodat het uittesten van de Unix software door een opvolger zal moeten gebeuren. Het is in dit verband dan ook van groot belang, dat de software logisch, gestructureerd, en voorzien van een ruime mate van commentaar is opgebouwd.
Ook is er aandacht besteed aan de ontwikkeling van software gereedschap. Dit gereedschap omvat onder meer utilities om automatisch alle gedefinieerde procedures, met de file naam waarin zij gedefinieerd zijn, te listen. Tevens vallen hieronder utilities ten behoeve van het automatisch genereren van listings en het maken van back-up's.
Alle inzichten die zijn verkregen bij de bestudering van de GKS software zijn in het verslag vastgelegd. De opgedane ervaring in grafische systemen, is gebruikt bij het vastleggen van het grafische systeem zoals dat in de toekomst zal moeten worden ontwikkeld. Tevens zijn de, in de toekomst nog te verrichten, werkzaamheden geinventariseerd en vastgelegd. De software listings zullen tesamen met het verslag en de richtlijn voor de toekomstige ontwikkelingen, dan ook een solide basis vormen voor de voortzetting van het werk aan de implementatie van GKS op het grafische systeem onder Unix.

12 Literatuuropgave

1 M.L. Verhoef,
Evaluatie en verdere uitbouw van een grafische module.
Afstudeerverslag THE Elektrotechniek, Eindhoven, 1984.

2 F.R.A. Hopgood, D.A. Duce en anderen,
Introduction to the Grafical Kernel System (GKS).
Academic Press, New York, 1983.

3 User Guide for C-GKS, the C Implementation of GKS, the Grafical Kernel System.
Centrum voor Wiskunde en Informatica, Amsterdam.

4 Installation Guide for C-GKS, the C Implementation of GKS, the Grafical Kernel System.
Centrum voor Wiskunde en Informatica, Amsterdam.

5 Unix Programmer's Manual, volume 1.
Bell Telephone Laboratories, Incorporated.
Holt Rinehart and Winston, New York, 1983.

6 Unix Programmer's Manual, volume 2.
Bell Telephone Laboratories, Incorporated.
Holt Rinehart and Winston, New York, 1983.

7 NEC Product Description Grafics Display Controller uPD 7220, versie 3/83 V3.1.
NEC Electronics (Europe) GMBH, Dusseldorf.