MOUNTE2

een Network Redirector

onder Ms-Dos voor het

Second Extended Filesystem

van Linux

A239
Ontwerp en Implementatie
van
Operating Systems

Student : Peter van Sebille
Studienummer : 756669
Begeleider : dr ir J. W. Mugge

Datum : juni 1995

Inhoudsopgave

1. Inleiding

Mounte2 betekent zoveel als: mount second extended filesystem. Met dit Dos progamma kunnen Linux partities gemound worden met een *second extended filesysteem* (in dit verslag kortweg ext2fs genoemd). Het resultaat is dat er een extra Dos schijf bij komt waar alle normale Dos acties en programma's transparant mee kunnen werken. In deze versie van mounte2 is het etx2fs alleen leesbaar[1].

Alvorens op de technische details van mounte2 in te gaan is het miscchien wel aardig om het hoe en waarom van mounte2 te vertellen.
Na het installeren van Linux, ongeveer 1½ geleden, wilde ik ook wel eens een Linux kernel project uitvoeren. Helaas ontbrak het mij aan tijd en aan originele ideeen om iets uit de grond te stampen. Na verloop van tijd kreeg ik het idee om een device driver onder Dos te schrijven die mijn ext2fs Linux partitie zichtbaar zou maken; ik had eindelijk een project. In eerste instantie was het puur hobyisme totdat ik me bedacht dat het wellicht een aardige opdracht voor dit vak zou zijn. Mijn begeleider stemde in met het opdrachtvoorstel en ik ging weer vrolijk verder met mijn device driver. Ik zat toen ongeveer op het punt dat ik enerzijds een dummy device driver had en anderzijds een stel basis routines waarme het ext2fs kon worden benaderen (alleen superblock lezen). Het was dus zaak om deze twee dingen te integreren tot een geheel. En daar zat 'm nou net het struikelblok want ik liep tegen allerlei beperkingen aan (zie hoofdstuk 4, Technische Oplossingen). Ik begon dus verwoed op ftp sites (met name *Simtel*) te zoeken naar voorbeeld programma's. Daar kwam ik een spannend programma tegen dat een alternatief bood voor *MSCDEX* (Microsoft CDrom Extentions). MSCDEX is in staat om cd-rom schijven transparant onder Dos benaderbaar te maken. Dit is hetzelfde mechanisme als mijn opdracht: maak een andersoortig file systeem dan het Dos FAT systeem zichtbaar onder Dos. Van het programma werd ik weining wijzer (assembler source), maar ik begreep wel dat dit de manier was waarop het eigenlijk zou moeten. Gelukkig stond er in de source een verwijzing naar een boek ([6]), dus ik dat vlug gekocht en met rode oortjes hoofdtuk 8 (The DOS Filesystem and Network Redirector) gelezen; zie voor een korte samenvatting hoofdstuk 4. Ik besloot direct om mijn device driver in de prullebak te gooien en me te concentreren op het maken van een *network redirector*. In [6] staat het voorbeeld programma *phantom* beschreven; dit implementeert een dynamic (un)loadable ramdisk als een network redirector. Mijn eerste "simpele" opdracht bestond dan ook uit het compileren van de bijgeleverde source van dit programma, welgeteld 1 .C file (> 2500 regels!). Met Borland C++ 2.1 ging dit niet, dus snel een Microsoft compiler geregeld. Nu was het voorbeeld programma met versie 6.0 van de deze compiler gecompileerd en ik had versie 7.0, mijn gecompileerde phantom werkte dan dus ook niet. Om de een of andere reden bleek een simpele *struct copy* de oorzaak van het probleem te zijn (zie source bijlage XXX, file redir.c, functie *redirector*). Na dit vervangen te hebben werkte het programma zowaar maar ik was wel drie dagen verder. Enfin, na deze tegenslag liep het eigenlijk wel vlotjes, al zaten er her en der nog wat valkuilen; met name een TSR met gealloceerd geheugen resident houden was

[1] Het is de bedoeling dat in de toekomst ook schrijven wordt ondersteunt maar dat ligt buiten de scope van deze opdracht

niet makkelijk. Het moet wel van het hart dat dit phantom programma nou niet bepaald uiblinkt in schoonheid (persoonlijke smaak!) : source in 1 file en dus allerlei componenten (tsr / xms / redirector / dos fat) door elkaar heen, cryptische namen voor variabelen (wie declareert er nou een globale variabele *r* ????). Nou ja, de rest was dan tenminste nog duidelijk.

Tot zover het “verhaal” achter MOUNTE2. Hieronder volgt nog een kort overzicht van wat nog komen gaat.
Het volgende hoofdstuk geeft de functionele beschrijving weer van deze opdracht. Hoofdstuk 3 geeft achtergrond informatie over partities en filesystemen. In hoofdstuk 4 worden van de verschillende technische oplossingen de voor- en nadelen opgenoemd en uitgelegd waarom voor de redirector is gekozen. Hoodstuk 5 geeft het technisch ontwerp van MOUNTE2, hoofdstuk 6 de implementatie details. In hoofdstuk 7 wordt het gebruik van MOUNTE2 kort toegelicht en de commandline parameters beschreven.. In hoofdtuk 8 worden een aantal conclusies gepresenteerd.

2. Functionele Beschrijving

Hieronder volgen het doel en de functionele eisen van de opdracht, zoals beschreven in de *taakopdracht*.

Doel: Het schrijven van een Dos device driver die het mogelijk maakt om transparant het second extended filesystem (ext2fs) van Linux te benaderen.De nadruk ligt meer op het schrijven van de device driver dan het doorgronden van het ext2fs.

Functionele eisen:

- de device driver moet het ext2fs kunnen lezen, schrijven valt buiten het kader van deze taak
- name clashes die ontstaan omdat het ext2fs namen tot 256 karakters ondersteunt en Dos maar 11 moeten "elegant" worden opgelost
- de driver moet op een willekeurige 386+ kunnen draaien (uiteraard met een ext2fs partitie)

Zoals al in de Inleiding is vermeld, is de driver vervangen door een *network redirector;* in het hoofdstuk Technische Oplossingen zullen we zien waarom. We kunnen zonder afbreuk te doen aan de inhoud van het doel an de functionele eisen, het begrip *device driver* vervangen door *network redirector*.

3. Partities en Filesystemen

Voordat we verschillende oplossingen kunnen bespreken volgt hieronder een uitleg over partities en filesystemen. We zullen zien hoe de partitie tabellen op een harde schijf gelezen en geinterpreteerd kunnen worden, vervolgens hoe filesystemen herkend en benaderd worden. Ten slotte volgt een globale bespreking over het Dos FAT filesysteem en UNIX filesystemen in het algemeen.

3.1 Partities

Op het allerlaagste niveau is een harde schijf opgebouwd uit *cylinders*, *tracks* en *heads*, het is echter makkelijker om een harde schijf te zien als een array van *sectoren*, dit zijn de kleinste data eenheden (512 bytes) op een harde schijf. Het totaal aantal sectoren is gelijk aan: aantal cylinder * aantal heads * aantal sectoren per track. Elke harde schijf is verdeeld in 1 of meer partities, op deze manier kunnen meerdere (verschillende) filesystemen op een harde schijf voorkomen. In eerste instantie zijn er 4 partities, dit zijn de zogenaamde *primary* partities. In de allereerste sector van een harde schijf (cylinder 0, head 0, sector 1) staat een partitie tabel met 4 entries. Elke entry beschrijft onder andere waar op de schijf een partitie begint, hoe lang deze is en welk type filesysteem op die partitie staat (een nummer). Voor een harde schijf met twee primary Dos partities groter dan 32 MB[2] zou er bijvoorbeeld als volguit uit kunnen zien:

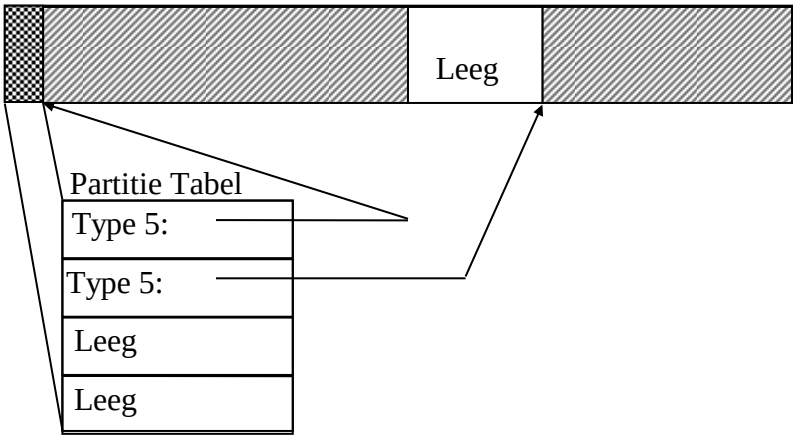

Figure 1: Harde schijf met twee primary partities.

Om een schijf in meer dan 4 partities te kunnen verdelen, kunnen *extended* partities worden aangemaakt. Een extended partitie heeft als type 6 in de entry in de partitie tabel. Binnen zo'n extended partitie kunnen dan meerdere *logische* partities worden aangemaakt. Aan het begin van *elke* logische partitie staat een partitietabel met twee entries; de eerste beschrijft op de normale

[2] Dit type filesteem heeft nummer 5 in de entry van de partitie tabel.

manier de logische partitie zelf (start, lengte, type etc) en de tweede entry is een link naar de volgende logische partitie binnen deze extended partitie.
Een harde schijf met een primary Dos partitie en een extended partitie met daarin twee Linux partities[3] zou er als volgt uit kunnen zien:

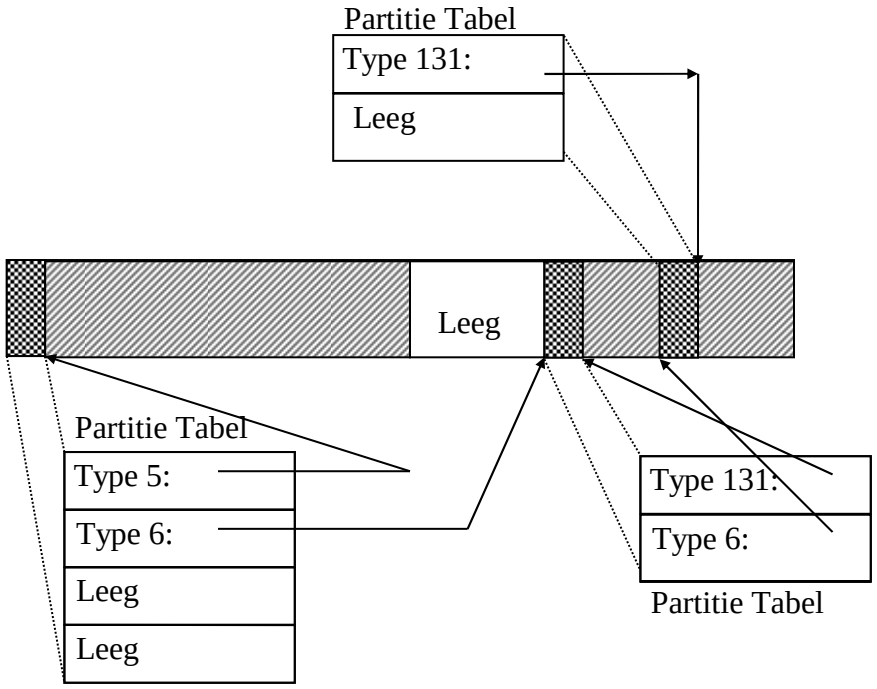

Figure 2: Harde schijf met een primary en een extended partitie.

Dos koppelt schijfletters op de volgende manier aan partities. Eerst worden de primary partities van alle harde schijven genomen, die krijgen een schijfletter in de volgorde waarvan ze gevonden worden, beginnend met letter c: op de eerste harde schijf. Vervolgens wordt verder gegaan met alle logische partities (binnen extended partitions) wederom beginnend op de eerste harde schijf. NB onder Dos kan nooit aan een hele harde schijf of een extended partitie in zijn geheel worden gerefereerd, alleen aan de afzonderlijke primary partities of de logische partities binnen een extended partitie.
Onder Linux is de benaming anders; er kan zowel aan een hele schijf, een primary partitie, een extended partitie of een logische partitie worden gerefereerd. De eerste harde schijf heet *hda*, de tweede *hdb* etc. De primary en extended partities op de eerste harde schijf heten dan *hda1*, *hda2*, *hda3* en *hda4*. De logische partities beginnen dan met 5 en worden op dezelfde manier doorlopen zoals Dos.
In Figure 2 zijn dan onder Linux (als dit de eerste harde schijf is) de volgende *devices* te gebruiken: hda (hele schijf), hda1 (Dos partitie), hda2 (extended partitie), hda5 en hda6 als twee logische partities.
Dos ondersteunt alleen zijn eigen FAT filesysteem en ziet op een harde schijf als in Figure 2 alleen de eerste partitie en noemt dat schijf c:.

[3] Linux gebruikt type 131 in de entry van de partitie tabel

3.2 FileSystemen

Filesystemen worden gebruikt om, hoe kan het ook anders, files te kunnen opslaan. De manier waarop files worden opgeslagen en worden terug gevonden is filesysteem afhankelijk. Er zijn veel verschillende filesystemen, we zullen ons dan ook alleen concentreren op de twee filesystemen die voor ons van belang zijn: het FAT filesysteem [4]van Dos en UNIX filesystemen in het algemeen. Alvorens op iets meer detail in te gaan, eerst nog een aantal belangrijke verschillen tussen beide soorten filesystemen. Dos kent eigenlijk maar een soort file (normale data files), terwijl UNIX naast deze normale files ook zogenaamde *speciale files* ondersteunt, te weten: *devices* (zowel block als charecter), *sockets, named pipes* (ook wel *fifo's* = first in first out, genoemd) en *symbolic links*. Verder zijn filenamen onder Dos beperkt tot 11 karakters, 8 voor de punt en 3 achter. De meeste UNIX filesystemen (zo ook ext2fs) ondersteunen namen tot 256 karkters. Het laatste onderscheid is dat Dos een typisch single user systeem is, terwijl UNIX een multi user systeem is. Het Dos file systeem kent dan ook geen protectie mechanisme op files, UNIX filesystemen wel in de vorm van toegangsrechten voor *owner*, *group* en *others*.

3.2.1 Dos FAT Filesysteem

Het Dos FAT filesysteem is een heel simpel filesysteem. Twee soorten dataeenheden zijn van belang: *sectoren* en *clusters*. Over sectoren hebben we het al gehad (512 bytes lang), clusters zijn de kleinst alloceerbare dataeenheid binnen het FAT filesysteem. Als een file of directory wordt gecreerd, neemt de data tenminste1 cluster in beslag, ook al is de file of directory leeg. Clusters bevatten een vast aantal sectoren afhankelijk van de grootte van de partitie en wordt tijdens formateren bepaald. Globaal ziet het FAT er als volgt uit.

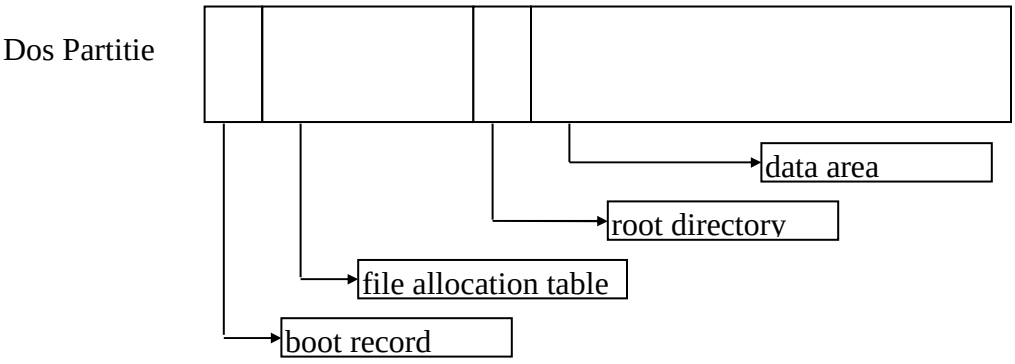

Figure 3: Globaal overzicht van het Dos FAT filesysteem.

Hieronder volgt een korte beschrijving van de componenten uit Figure 3:

[4] We beschouwen alleen de FAT16 filesystemen die lengtes groter dan 32MB ondersteunen.

- boot record: hierin staan de parameters van de het filesysteem, zoals het net genoemde aantal sectoren per cluster, de lengte van de file allocation table en het aantal entries in de root directory.
- file allocation table: dit is een tabel met evenzoveel 16 bits entries (daarom wordt dit filesysteem ook wel eens FAT16 genoemd[5]) als dat er clusters zijn. Deze tabel wordt gebruikt om bij te houden welke clusters er vrij zijn, welke clusters een *bad sector* bevatten (en dus onbruikbaar zijn) en welke clusters tot welke file of directory behoren.
- root directory: hier staan alle *directory entries* behorende bij de root directory; deze krijgt tijdens het formateren een vast aantal entries toegewezen dat niet meer veranderd kan worden.
- data area: dit de ruimte in het file systeem waar subdirectories en files worden opgeslagen.

Directories worden geadministreerd met een 32 bytes lange *directory entry*, zoals weergeven in Table 1.

lengte (bytes)	**Betekenis**
8	Filenaam
3	Extensie
2	Attribuut
9	Gereserveerd
2	Tijdstip van creatie of wijziging
2	Datum van creatie of wijziging
2	Begin cluster
4	Grootte in bytes

Table 1: Formaat van een Dos directory entry.

Als een file wordt gecreerd, wordt in de directory entry diens begin cluster gezet. Met dit cluster nummer kan in de file allocation table alle ander clusters worden gevonden die bij deze file horen. Als het clusternummer *n* is, staat in de *n*-de entry het volgende cluster nummer, op de offset van dat cluster nummer de volgende etc. Directories zijn eigenlijk files waarvan de data uit directory entries bestaat, ze worden op dezelfde manier behandeld (wat betreft zoeken, aanmaken en verwijderen) als files. Aan een bit in het attribuutveld van een directory entry kan Dos zien of het om een file, directory, hidden file, system file of volume naam gaat.
Omdat de file allocation table16 bits entries bevat, kunnen er ten hoogste ongveer 64K clusters zijn.

3.2.2 UNIX FileSystemen

[5] Er bestaat ook een FAT12 systeem met 12 bits entries; het aantal clusters dan ook beperkt tot 2^12.

UNIX filestystemen zijn er in vele smaakjes, de basis is vaak hetzelfde en de verschillen zitten in de manier waarop bepaalde informatie wordt geadministreerd. De kleinst mogelijke alloceerbare eenheiden heten *blocks*. De *blocksize* is meestal in de orde van grootte van 1 KB. Het ext2fs kan geformateerd worden met een blocksize van 1,2 of 4KB.
Zoals al eerder gezegd ondersteunen UNIX filesystemen behalve directories en files ook speciale files. De gemeenschapplijke kreet voor dit alles is *node*, en met elke node wordt een *inode* (= indexed node) geassocieerd. Een inode bevat alle informatie over de node en behelst onder ander het volgende (precieze invulling is filesysteem afhankelijk), zie Figure 4

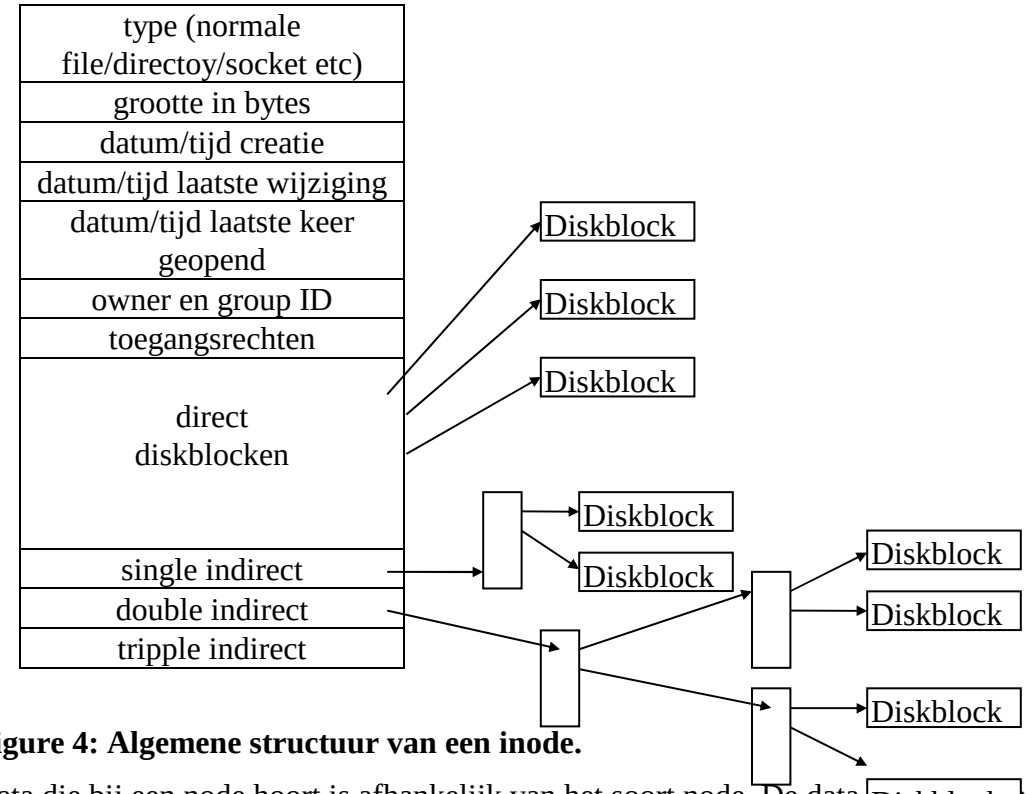

Figure 4: Algemene structuur van een inode.

De data die bij een node hoort is afhankelijk van het soort node. De data bij een file en directories staan daadwerkijk op disk; die van de speciale files meestal niet (symbolic links weer wel). Nodes voor speciale files worden gebruikt om allerlei objecten zoals speciale hardware (via devices) of netwerkverbindingen (via sockets) op dezelfde manier te benaderen als gewone files. Zoals te zien is in bovenstaande figuur wordt gebruikt gemaakt van indexering om de datablocken die bij een node horen kunnen vinden. De directory entries van UNIX file systemen zien er als volgt uit, zie Figure 5.

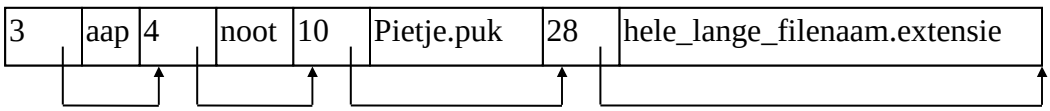

Figure 5: Algemene structuur van directory entries.

Directory entries bevatten over het algemeen alleen de naam van de node, het inode nummer en lengte van de naam.

Als laatste moet nog vermeld worden het *superblock.* Dit is het eerste datablock op een UNIX filesysteem en bevat de parameters van het file systeem zoals de blocksize en het aantal indodes. De verschillen tussen afzonderlijke UNIX filesystemen zitten voornamelijk in de manier waarop inodes en datablocken op schijf worden opgeslagen en hoe een nieuwe inodes of datablocken kunnen worden gealloceerd. Vaak worden allerlei (ingewikkelde) datastrcuturen bijgehouden om fragmentatie tegen te gaan en om snel inodes en diskblocken terug te vinden. Helaas is het ext2fs niet zo goed gedocumenteerd en is mij de filosofie achter de allocatie routines niet echt duidelijk.

4. Technische Oplossingen

Er zijn drie manier in Dos om filesystemen anders dan het Dos FAT te ondersteunen: *front-end hooks*, *device drivers*, *back-end hooks* (=network redirector).

Voordat we deze drie manier kunnen beschrijven, bekijken we eerst hoe file operaties door Dos worden geinterpreteerd en uitgevoerd, zie Figure 6.

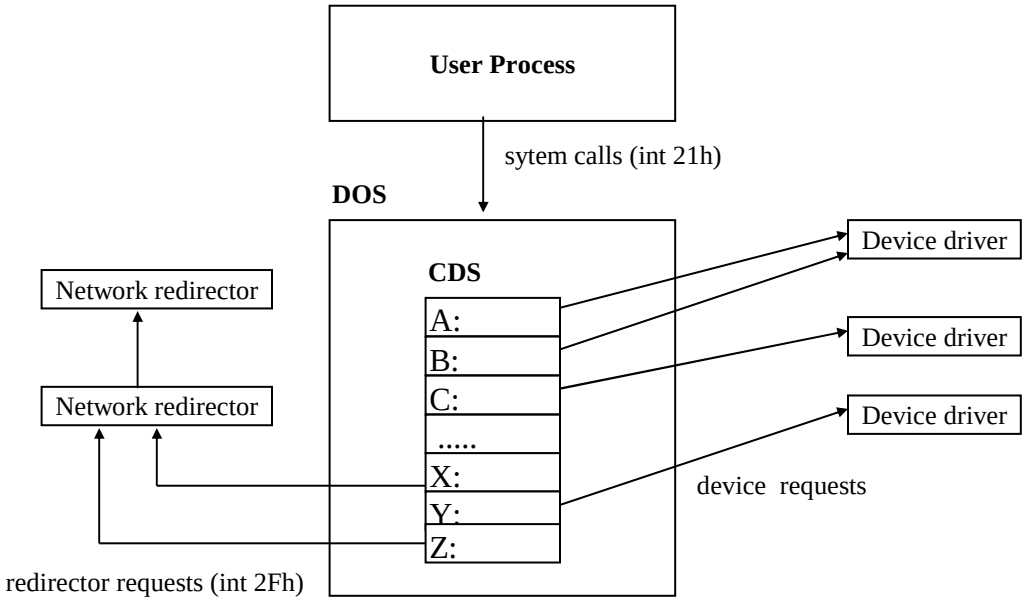

Figure 6: File operations in Dos.

Om file operaties (zoals *cd*, *open*, *close*, *read*, *write*, etc) te kunnen uitvoeren, roept een user process system calls van Dos aan. *Interrupt 21h* is daar een van de belangrijkste van (zeker wat file operaties betreft). Dos zal bij een system call die betrekking heeft op een file eerst kijken op welke schijf deze operatie moet worden uitgevoerd. Dos houdt intern een datastructuur bij genaamd CDS (=Current Directory Structure) bij, die niet alleen gebruikt wordt (zoals de naam al doet vermoeden) om de huidige directory van een schijf in op te slaan, maar die ook aangeeft wat voor soort schijf dit is. Er zijn evenveel entries in de CDS als dat er schijven zijn gedefinieerd, dit aantal kan gestuurd worden door het *lastdrive* commando in config.sys. Het soort schijf kan zijn: *physical*, *network*, *cdrom*, *subst* of *join*. We zullen het type cdrom, subst en join hier buiten beschouwing laten en ons concentreren op physical en network schijven.
Als het schijf van het type phyical is, dan zal Dos *device requests* genereren voor de bijbehorende device driver; dit is overigens altijd een block device driver. Een belangrijke constatering is dat Dos ten alle tijde een Dos FAT filesysteem op een block device wil zien. Dit betekent dat Dos zelf de vertaling maakt van file operaties naar cluster en sector nummers. De belangrijkste routines die een block device driver moet implementeren zijn dan ook: *read sector* en *write sector*.

In tegenstelling tot physical schijven, maakt Dos maakt geen enkele aanname over het soort filesysteem die op network schijven staan. Elke file operatie op een network schijf wordt door Dos dan ook doorgestuurd naar een *network redirector* via *interrupt 2Fh* (subfunctie 11h). Deze network redirector moet de feitelijke file operatie uitvoeren en het resultaat van deze aktie aan Dos teruggeven. De interface tussen Dos en network redirectors komt in grote lijnen overeen met de Dos system calls voor file operaties, dwz dat voor de meeste system calls voor file operaties (int 21h) er een soortgelijke call bestaat richting network redirector (int 2Fh). Omdat er meer dan 1 network redirector aanwezig kan zijn, vormen alle int2fh interrupt handlers van network redirectors een *chain*. Elke redirector is zelf verantwoordelijk om te beslissen of het binnen gekomen request voor hem bedoeld is of dat het doorgestuurd moet worden naar de volgende redirector in de *chain*.

Na deze uitleg kunnen we de drie manieren om andersoortige filesystemen te ondersteunen binnen Dos, nader bespreken

Front-end Hooks:
Een front-end hook zal interrupt 21h afhangen om system calls op file operaties te kunnen filteren, nog voordat Dos die ziet. Op deze manier kan zo'n front-end hook een willekeurig filesysteem implementeren. Het grote voordeel is dat alles mogelijk is, ook om bv lange filenamen (groter dan 11) te ondersteunen. Een belangrijk nadeel, om op deze manier andersoortige filesystemen te ondersteunen, is dat *elk* system call dat betrekking heeft op files zal moeten worden vervangen door een eigen versie. Deze eis impliceert dat het *exec* system call ook moet worden vervangen en dat is meer dan alleen een vervelende bijkomstigheid.

Device Drivers:
In eerste instantie (zie ook hoofdstuk1) was het de bedoeling dat ik op deze manier de opdarcht zou uitvoeren, helaas heeft deze manier nogal wat nadelen. Zoals al eerder is opgemerkt wil Dos op een block device het Dos FAT filesysteem zien. Bij het interpreteren van file operaties op een block device zal Dos zelf de vertaling maken van symbolische file- en directorynamen naar clusters en sectoren. Als een programma b.v. een *dir* commando in de root directory van zo'n block device uitvoerd, zal Dos eerst het cluster nummer berekenen van de root directory op het device en vervolgens de device driver een aantal requests sturen om sectoren te lezen. In ons geval, waar we het ext2fs willen ondersteunen, zal de device driver de *read sector* requests moeten interpreteren en nagaan wat Dos eigenlijk verwacht te zien in deze sectoren. Bij het bovenstaande *dir* commando in de root directory, zal de device driver de root directory van het ext2fs lezen en de gevonden entries *vertalen* naar Dos directory entries. En hier zit 'm nu net de kneep. Met de informatie die Dos uit deze sectoren terug krijgt (directory entries) ziet Dos telkens een stukje meer van het filesysteem. De directory entries vertellen Dos namelijk waar op het device de subdirectories en de files in deze directory zich bevinden (het eerste clusternummer van een file of directory). Dit laat zich het best illustereren met het volgende voorbeeld: een cd naar \usr\local\bin. Dos zal wederom eerst het cluster nummer berekenen dat bij de root directory hoort en zal de device driver verzoeken de bijbehorende sectoren te lezen. Deze sectoren bevatten de directory entries in de root directory en Dos zal op zoek gaan naar een entry met de naam *usr*. Als deze entry is gevonden, gebruikt Dos het start cluster nummer uit de entry om de directory entries voor van *usr* te vinden. De device driver krijgt wederom vezoeken binnen om

sectoren te lezen. In deze sectoren gaat Dos op zoek naar de entry *local*. Op dezelfde manier wordt nu in directory *local* naar entry *bin* gezocht en als deze entry een directory is, is de cd operatie geslaagd. Waarschijnlijk wordt het beeld al duidelijk waar hier de moeilijkheid zit: Voor *alle* directory entries die we aan Dos teruggeven zullen we moeten bijhouden op welk cluster nummer de data begint die bij deze entry hoort (file of subdirectory). Bovendien bestaan files en directories uit meer dan een cluster, dus moeten we ook in de file allocation tabel aangeven waar de overige clusters staan.
Nog voordat we ingaan of dit wel haalbaar, hoe we dit oplossen en wat dit voor de performance betekent, kunnen we al direct constateren dat we op deze manier *nooit* het ext2fs kunnen schrijven. Immers, als Dos onze device driver verzoekt om sector *y* te schrijven, kunnen we met alleen dit sector nummer nooit achterhalen wat de achterliggende gedachte is van dit verzoek; wordt er een directory aangemaakt ? een file weggeschreven? een nieuwe file aangemaakt of directory gecreeert ? en zo ja in welke directory gebeurde dat? Dit zijn allemaal vragen die we met alleen een device driver niet kunnen beantwoorden; we zouden dan ook een soort *front-end monitor* moeten schrijven die in de gaten houdt met welk system call we bezig zijn om op die manier de *context* van een schrijf verzoek te achterhalen.
Genoeg van dit alles. Laten we maar eerst kijken naar de derde en laatste manier om “vreemde” filesystemen te ondersteunen.

Back-end Hooks (network redirector):
Back-end hooks zijn ideaal om andersoortige filesystemen te ondersteunen. Sterker nog, ze zijn er voor bedoeld. Eigenlijk zijn er maar een paar nadelen. De belangrijkste is dat network redirectors niet officieel gedocumenteerd zijn door Microsoft[6]. Er zijn volgens [6] wel degelijk sterke aanwijzingen dat bepaalde fabrikanten deze documentatie wel hebben gekregen van Microsoft. Overigens is MSCDEX van Microsoft ook op deze manier gebaseerd. Een ander nadeel is dat network redirectors toegang moeten hebben (zowel lezen als schrijven) tot elementaire datastructuren van Dos (zoals de CDS uit Figure 6). De paradox is dat Dos dit zelf impliciet toestaat, er is namelijk een (ongedocumenteerde) system call (int 21h/ sub 5200h), genaamd Get*LOL* (= get List of Lists), die een pointer retouneert naar een tabel met pointers naar Dos datastructuren, waaronder de CDS. Bovendien verwacht Dos dat redirectors zelf allerlei tabllen bijwerken, dit zit in de hele int2F/sub 11h interface verweven. Een redirector moet b.v. zelf de huidige directory in de CDS bijwerken na een *cd* call. De datastructuren waar het hier om gaat zijn:

- LOL: De List if Lists met onder andere een pointer naar de CDS
- SDA: De Swappable Dos Area met allerlei “kladvariablen” zoals een seachrecord en directory entry voor *ffirst/fnext* calls en filenaam (inclusief pad) voor *open* calls en een tweede filenaam voor calls met twee filenamen als parameters (*delete/rename*). De SDA wordt verkregen met de (ongedocumenteerde) system call int21h/sub 5D06h
- SFT: System file table. Deze tabel bevat de informatie over geopende files. Dos geeft (via registers) een pointer naar een entry in de SFT door bij sommige int2Fh calls (zoals *open/read/write*). De redirector kan met deze pointer informatie ophalen en bijwerken in de desbetreffende entry, b.v. de filepointer.

[6] Network redirectors zijn in [6] beschreven.

Een ander nadeel is dat niet alleen alleen redirectors in de int2F interrupt chain hangen, maar ook allerlei ander programma's die dit interrupt gebruiken (andere subfuncties). Dit komt de performance uiteraard niet ten goede. Voor een volledige beschrijving van de interface tussen Dos en network redirectors wordt verwezen naar [6]. We zullen ons hier, om een globale indruk te geven, beperken tot een opsomming van de belangrijkste calls: remove directory, make directory, change directory, close file, commit file, read file, write file, set file attributes, get file attributes, rename file, delete file, open file, create file, find first, find next en seek from end.

Conclusie:
Het is duidelijk dat network redirectors de manier is om andersoortige filesystemen te ondersteunen. Van alle mogelijke nadelen van het gebruik van network redirectors is eigenlijk alleen het feit dat het een ongedocumenteerde feature is van belang. Dit verslag is echter niet de juiste plek om deze discussie te voeren. We zullen dit nadeel dan ook naast ons neerleggen en ons ontwerp baseren op een network redirector.

5. Technisch Ontwerp

Hieronder staan de belangrijkste ontwerp beslissingen van de network redirector:

- Schrijven: We zullen ons in eerste instantie beperken tot het leesbaar maken van een ext2fs partitie. Schrijven is uiteraard niet minder belangrijk maar er zijn twee argumenten om daar nu van af te zien. Schrijven naar een ext2fs is substantieel moeilijker dan lezen. In pargraaf 3.2.2 is al uitgelegd dat UNIX filesystemen zich voornamelijk onderscheiden in de allocatie routines, die vaak moeilijk te begrijpen zijn; het extfs2 is daar geen uitzondering op, er is trouwens ook geen documentatie alleen de kernel sources. Het tweede argument is dat het kunnen schrijven naar ext2fs goed getest moet worden, in ieder geval meer en beter dan voor lezen. Beide punten zouden erg veel tijd kosten en dat betekent dat waarschijnlijk half of slecht werk, twee dingen waar ik niet van houd.
- TSR: We implementeren onze network redirector als een TSR (Terminate and Stay Resident). Ik zou touwens niet eens weten hoe het anders zou moeten.
- Filenamen:
    - Speciale tekens in filennamen:
        - De volgende tekens in de filenamen van het ext2fs worden geconverteerd naar het teken _ (underscore): *?<>\$+=;
        - De . (dot) van hidden files van het ext2fs wordt ook geconverteerd naar een _ (underscore)
        - De eerste . (dot) wordt beschouwd als het begin van de file extensie, alle dots die volgen worden ook geconverteerd naar een _ (underscore)
    - Lange filenamen: Om te lange filenamen (een filenaam is te lang als het meer dan 8 karakters voor de punt heeft of meer dan drie achter de punt) onder Dos uniek te houden wordt het volgende algoritme gebruikt:
        1. kap eerst af, d.w.z. neen voor de punt ten hoogst 8 en na de punt ten hoogste 3 karakters.
        2. Om de file uniek te maken wordt de offset van de file in de directory genomen, het is vrij aannemlijk dat dit een getal is tussen 000-999. Gebruik deze drie-letterige offset als filenaame extensie.

        Deze conversie is niet waterdicht maar heeft het voordeel dat namen niet ongeloofelijk cryptisch worden. Overigens zegt, over het algemeen, het eerste karakter van de file extensie iets over het type van de file (.c .o .a etc). Om het bovenstaande algoritme te verfijnen staan we toe dat de gebruiker kan aangeven (comand line optie) met hoeveel karakters van de drie-letterige offset de filenaam uniek gemaakt moet worden.
- We zullen een cache in expanded memory bijhouden voor diskblocken en inodes. Waarschijnlijk zal *smartdrive* ook wel in staat zijn om onze diskblocken te bufferen, maar dit wilde ik al heel lang eens doen (schrijven van een cache); het gaat er tenslotte toch ook om dat je er wat van leert, niet?
- Ik ben nogal onder de indruk van de hele VFS (Virtual FileSystem) laag in de meeste UNIX systemen (ook Linux). We zullen ook een VFS laag in te bouwen in onze redirector om

zoveel mogelijk de specifieke eigenschappen van het ext2fs te verbergen voor de redirector. Op deze manier zou in de toekomst een soort MSCDEX achtige redirector kunnen worden gemaakt. MSCDEX communiceert met cd-rom spelers via een *character device driver* die fabrikanten kunnen schrijven voor hun cd-rom spelers. Op dezelfde wijze zou onze redirector kunnen communiceren met character device drivers (of via een zelf gedefinieerde interface) die de afzonderlijke file systemen aansturen.

- Speciale files: Onder Dos hebben de UNIX speciale files sockets, devcices en named pipes geen betekenis, de normale files, directories en symbolic links wel. De eerste drie speciale files laten we *zien* als normale files met als inhoud een regel als “Dit is een xxxx file”, met xxx het type file. Normale files en directories kunnen 1 op 1 vertaald worden naar Dos files en directories. De symbolic links wordt in eerste instantie behandeld zoals sockets,devices en named pipes; in de toekomst moeten symbolic links worden “gevolgd”, maar dit valt buiten deze opdracht.
- Orginele filenamem: Zoals in een van de vorige punten is besproken worden de filenamen van het ext2fs geconverteerd naar Dos namen. We zullen in *elke* directory een *phantom* file creeeren, die dus niet fysiek op disk staat, waarin tenminste de alle files staan met hun originele en hun gevonverteerde naam.
- Omdat een TSR niet onder een debugger kan worden gedraait, zal een uitgebreidde debug logging mechanisme worden ingebouwd. Aangezien we vanuit een interrupt handler niet naar files kunnen schrijven, en het ook niet wenselijk is om debug messages op het beeldscherm te schrijven, wordt gelogd naar expanded memory. Allerlei utilities, waaronder het programma *ems.com* van de fabrikant Quaterdeck, kunnen willekeurige expanded memory pages naar file schrijven. Uitermate handig!
- Een end-of-line in een textfile onder UNIX systemen bestaat alleen uit het karakter 13 (carriage return), onder Dos is dat de combinatie van karakter 10 (line feed) en 13 (carriage return). Naive Dos programma’s zullen dan ook geen end-of-line karakters detecteren in een UNIX textfile. We zouden een conversie slag kunnen uitvoeren bij het lezen van textfiles om extra line feeds toe te voegen (zoals gebeurt onder Linux bij het mounten van Dos filesystemen). Het vervelende is dat we niet weten welke files van het ext2fs textfiles zijn en welke niet (onder Linux is het mechanisme om te bepalen welke Dos file textfiles zijn en welke niet ook niet waterdicht en is gebaseerd op de extensie van files). We zullen zo’n conversie slag dan ook niet uitvoeren, zeker omdat er genoeg utilities onder Dos draaien die UNIX textfiles wel goed interpreteren (b.v. *vim*, een Dos clone van *vi*).

Onze redirector heeft dan de volgende modulaire structuur, zie Figure 7.

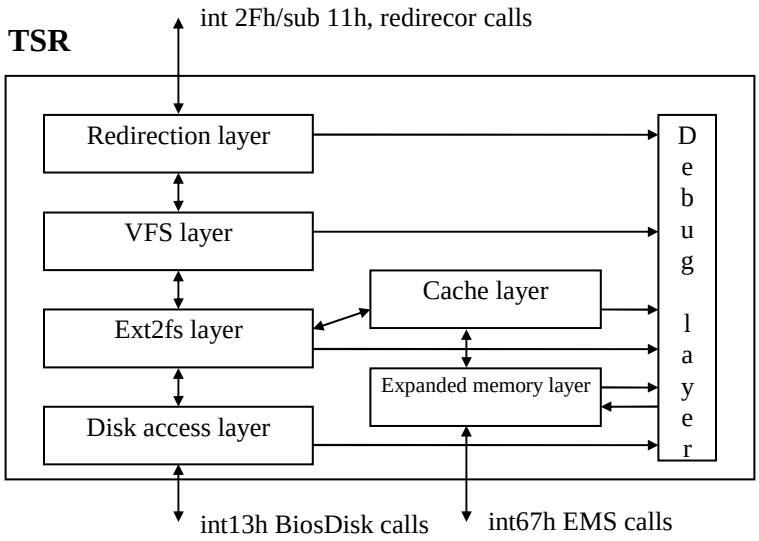

Figure 7: Structuur van onze redirector.

6. Implementatie

We zullen enige implementatie details bespreken van onze redirector, die ik zoals al eerder gezegd *mounte2* heb genoemd. De beschrijving van de afzonderlijke componenten komen grotendeels overeen met de modules zoals geschetst in Figure 7 in het vorige hoofdstuk.

Mounte2 is met Microsoft C 7.0 gecompileerd in het *small model*. Op *tsr.c* en *redir.c* na kunnen alle .c files ook met Borland C/C++ (vanaf in ieder geval 2.0) gecompileerd worden. Dit laatste is handig omdat een heleboel spul dan getest kan worden onder de Borland Debugger (zelf even een dummy *main()* schrijven). Verder is het geheel strak modulair opgezet, d.w.z., elke module wordt geimplementeerd in 1 .c file en interfacing gebeurt via header files. Dit heeft als voordeel dat modules op zichzelf staan, tenzij ze uiteraard "steunen" op lager gelegen modules. De .c files zijn als volgt gecomplieerd[7]:

```
cl -c /W3 /Gs /Os xxxxx.c
```

en zijn verder zonder speciale vlaggen gelinkt tot 1executable.

Omdat alle *struct* definities van alle ext2fs datastructuren (inode, superblock, directory entries, etc) stukken van mounte2 bijna letterlijk van de Linux source zijn gekopieerd, wordt gebruikt gemaakt van de volgende *typedefs*: uint8, int8, uint16, int16, uint32, int32, omdat een *int* onder Linux 32 bits lang is en onder Dos 16 bits. Daar waar de lengte van een *int* belangrijk is, worden deze type definities gebruikt.

We zullen nu de afzonderlijke modules bekijken (van laag naar hoog).

biosdisk:
Dit implementeert de routines om via het bios fysieke sectoren te *lezen* en te *schrijven* en om de *parameters* van een harde schijf uit te lezen.
Opmerking: Bios call GetDriveParam (int13h/sub 08h) retourneert het maximale aantal *cylinders, sectoren* per track en *koppen*. Het is me een raadesl waarom het aantal koppen dat deze call terug geeft 1 lager is dan dit maximum. In zowel [1], [3] en [7] wordt hier met geen woord over gerept. Vreemd!

diskdev:
Hier wordt het concept *fysieke* en *logische* devices geimplementeerd. Fysieke devices komen overeen met Linux' hda, hdb etc, terwijl logische devices overeenkomen met de afzonderlijke partities, dus hda1, hda2 etc. Fysieke en logische devices hebben *read* and *write* routines. Tevens wordt de basis gelegd voor het feit dat er op logische devices filesystemen staan. Voor een logisch device kan een *blokgrootte* worden opgegeven en hiermee kunnen dan dus *blokken* ipv *sectoren* gelezen worden (een sector is altijd 512 bytes, een blok is een aantal sectoren). Verder staan hier de routines om de partitie tabellen uit te lezen.

[7] De compileer vlaggen leveren de kleinst mogelijke code op.

ems.c:
Algemene module voor EMS 4.0 (Expanded Memory Specification). Niet alle routines zijn geimplementeerd, alleen de belangrijkste. De keuze voor EMS i.p.v. XMS (eXtended Memory Specification) is dat het voordeel van EMS is dat we een 64 KB buffer hebben, namelijk de *pageframe*.
Opmerking: Er zijn drie copieer routines: van conventional naar expanded memory en v.v., en van expanded naar expanded memory. Het is frapant dat een kopieer actie van data uit het pageframe (gemapt op een expanded memory pagina) naar expanded memory onder Ms-Windows alleen als een copieer actie van expanded naar expanded memory mag worden gedaan, terwijl onder Dos (Qemm 6.x) dit ook mag als copieren van conventional naar expanded memory (wat je toch eigenlijk zou verwachten).

vsprintf.c:
Dit implementeerd *simple_vsprintf,* welke wordt gebruikt in de debug module om geformatterde debug statements te schrijven. Het is een port van de Linux kernel source (er zat trouwens een bug in), met de aanpassing dat behalve *%s* ook *%fs* gebruikt kan worden om strings te printen. Dit laatste is nodig omdat een *char* * in het small model een near pointer is terwijl we heel wat *char far* * vanuit onze TSR willen printen (allerlei namen uit Dos datastructuren en uit de cache). We hadden ook de normale vsprintf uit de C-library kunnen nemen maar die groter (ook support voor floats ed) en hier kunnen we geen *char far pointers* mee afdrukken.

debug.c:
Dit heeft als belangrijkste routine: *dprintf(int flag,char *format,...)* die gebruikt wordt om debug statements te kunnen gebruiken.*Flag* is bitmask die een geeft om welke module het gaat, voor elke module is er een zo'n flag gedefinieerd. *Format* is een format string die grotendeels overeent komt met de format string van *printf* uit de standaard C-library, alleen de floats worden niet ondersteund. Voor de rest kan worden opgegeven of er naar *console* of naar *ems* moet worden gelogd. Over het algemeen heeft ems de voorkeur, maar in bepaalde gevallen (als-ie hangt) en er geen mogelijkheid meer is om de expanded pagina's op tijd te bewaren, is loggen naar console uitermate handig. Aan de *ems handle* die voor het debuggen wordt gebruikt, wordt de naam MOUNTE2 toegekend. Dit maakt het het zoeken naar de de goede handle wat makkelijker. Bovendien kan worden opgegeven (command line optie) of de handle moet worden vrijgeven of niet nadat mounte2 wordt *ge-unload.* Dit laatste is met name weer handig om allerlei *shutdown messages* nog te kunnen zien. Om niet onnodig elke keer deze pagina's te bewaren wordt bij het initialiseren gekeken of er niet al een ems handle bestaat met de naam MOUNTE2, zo ja dan wordt die genomen, anders worden nieuwe pagina's gealloceerd.

emscache.c:
Deze module implemeteerd een *two-level* cache zoals die ook onder Linux wordt gebruikt om directory entries te cachen. Dit is geen port of copy maar een eigen versie, van scratch geschreven. Deze module is zo opgezet dat meerdere cache pools kunnen worden gebruikt; elke pool cachet zo zijn eigen objecten (alle objecten binnen een pool hebben dezelfde grootte). De objecten zelf bevinden zich binnen het expanded memory, de datastructuur om een pool te beheren ligt in expanded memory. Deze datastructuur implementeert 1 lijst met twee

dubbelgelinkte ketens, 1 *last recently used* keten en 1voor de *hash list* (om een object weer snel op te zoeken). Het begrip two-level cache slaat op het feit dat er twee levels zijn. Nieuwe objecten worden in level 1 lru keten geplaatst en het meest oude object verdijnt uit de lijst. Een *cache hit* van een object uit level 1 betekent dat het object promoveert naar level 2 en het meest oude object uit level 2 degradeert naar level 1(waar het wel als most recent wordt gezet). Een cache hit in level 2 maakt dat object most recent. Er is voor zowel inodes als voor diskblokken een cache pool. Het enige nadeel aan deze module is dat de voor beide pools de 2-dubbelgelinkt lijsten zich in het conventional memory bevinden. Een entry in deze list is 22 bytes lang; om b.v. 500 diskblocken van 1KB te cachen is dan, naast 512KB expanded memory, 11 KB conventional memory nodig. Dit tikt aan in een *tsr* die maar 64 KB groot mag zijn (small model).

tsr.c:
Deze routines zijn nodig om een tsr te kunnen schrijven. Net als het *phantom* voorbeeld uit [6], is mounte2 dynamisch te loaden en te unloaden. Dit unloaden gebeurt als volgt: tijdens het loaden, vlak voor resident gaan van het programma, wordt de *PSP* (Program Segment Prefix) van het programma opgeslagen in een zogenaamd *signature record*. Binnen dit signature record wordt ook een string opgeslagen met de naam van het programma, dit wordt later gebruikte als *signature*. Vervolgens wordt er in de interrupt tabel een vrij interrupt gezocht in de range van 60h tot 67h; deze entry wordt gebruikt om het address naar het signature record op te slaan. Hierna “gaat” het programma resident. Om deze tsr te kunnen omloaden gaat het unload programma in de interupt tabel op zoek naar een entry waarvan de inhoud wijst naar het signature record van de draaiende tsr; dit gbeurt door op zoek te gaan de naar siganture string in het signature record. Is het signature record gevonden, dan kan met behulp van het PSP (van het tsr) in dat record de tsr worden ge-unload op de volgende manier:

1. Het unload programma zet het parent PSP van het draaiende tsr progamma op het PSP van het unload programma (zichzelf dus).
2. Het programma termination address van het draaiende tsr programma wordt op een address gezet binnen het unload programma.
3. Het PSP van het draaiende tsr programma wordt met het (ongedocumenteerde) system call s*etPSP* (int21h/sub 50h) als current gezet.
4. Het unload programma roept vervolgens het system call *Terminate* (int 21h/ sub 4ch) aan. Dos zal het huidig programma beeindigen en aangezien we in de vorige stap het draaiende tsr programma als huidig programma hebben gezet, wordt het tsr door Dos beindigt en Dos laat het beeindigde programma terugkeren naar het *program termination address* van het parent process. Vanwege stap 1 en 2 wordt terug gesprongen naar het unload programma.

In de praktijk is het unload mechanisme een onderdeel van de tsr zelf. Met een parameter (-U of iets in die geest) kan een tweede *instantie* van de tsr een instantie vorige unloaden. Ik heb deze tsr module flexibel opgezet en kan als raamwerk dienen voor elk ander tsr (het staat ook geheel op zich zelf). De volgende punten zijn nieuw:

1. Om de tsr *logica* geheel los te koppelen van de rest van een programma, is het signature record voor een deel vrij invulbaar (met b.v. pointers naar eigen data structuren). De tsr module hoeft niets van deze inhoud te weten.

2. Het unloaden gebeurt door een routine uit de tsr module aan te roepen, als parameter kan een callback functie worden meegeven. Deze functie wordt door de unload routine aangeroepen met als parameter het address van het gevonden signature record.
3. Het meest lastige van het het schrijven van een tsr is om in een C programma tijdens *runtime* te bepalen hoeveel geheugen de tsr in beslag neemt. De grootte van dit geheugengebied is het verschil tussen het *hoogste* en het *laagste* geheugen adres; het laagste adres wordt m.b.v. het PSP gevonden maar de truc is nu om het hoogste te vinden. Er staat in [1] geen enkel bruikbare algemene oplossing voor het geval de tsr een aantal *malloc's* uitvoert (zoals mounte2 voor de emscache's bv). In het geval dat een tsr niets alloceert is de einde van het *data segment* het hoogste geheugenadres. Bij C programma's geschreven in Microsoft C kan dit worden opgevraagd door het adres van variable *end* te nemen. Microsoft genereert op een dusdanige manier code[8] dat deze variabele op het einde van het data segment staat. Mijn oplossing bestaat uit de routines *tsr_malloc* en *tsr_calloc* die niet alleen de normale malloc en calloc aanroepen, maar ook bijhouden wat het hoogste adres is dat op deze manier wordt gealloceerd. Alle andere modules die onderdeel zijn van het tsr programma moeten dan tsr_malloc en tsr_calloc gebruiken om geheugen te alloceren.

ext2_fs.c:
Hier valt eigenlijk weinig speciaals over te vertellen. Dit is de module die inodes en blocken van de extt2_fs leest. Onze tegenhanger van de Linux structure *bufferhead* is *emscache_t*, bedoeld om blocken te buffereen (cachen). Er zijn 3 belanrijke routines: *ext2_readinode*, *ext2_readblock* en *ext2_bmap*. De eerste twee retourneren een *emscache_t *,* waarvan het veld *data* wijst naar het inode resp. diskblock. Na gebruik moet de verkregen emscache_t * worden vrijgeven (niet de cache inhoud in ems, maar de mapping in conventional memory) met *ext2_releaseinode* en *ext2_releaseblock*. De derde routine (blockmap) maakt de vertalingen van logische fileblocken naar fysiek diskblocken (zie de beschrijving van indexering van diskblocken in paragraaf 3.2.2).

vfs.c:
Het virtual filesystem verbergt de specifieke ext2fs details voor de redirector laag en zadelt van de andere kant het ext2fs niet op met Dos details. De belangrijkste routine is het lezen van diectory entries van een ext2fs directory: *ext2_entrylookup*. Dos zoekt met een *findfirst, findnext* mechanisme een directory af. Het vervelende is dat Dos alleen de offset van de laats bekeken directory entry hoeft bij te houden, aangezien alle directory enties dezelfde lengte hebben. Om niet onnodig vaak een ext2fs directory te herlezen (op zoek naar de directory entry *offset),* wordt een extra zoek structuur geintroduceerd voor het ext2fs (kan voor elk UNIX achtig filessyteem gebruikt worden) waarin van de huidige findfirst/findnext zoekactie het inode nummer van de huidige directory, het inode nummer van de laast gevonden entry in die directory, het laatste gelezen blocknummer, de offset binnen dat block en het totaal aantal gelezen entries bijgehouden.
Het vfs zorgt ook dat de speciale files (symbolic links, sockets, devices en pipes) worden "vertaald" naar geowne Dos files en mapt de inhoudt van die files op een text string ("dit is een socket", voor sockets bv.). Als laatste creeert het vfs in elke directory een *phantom* file, waar de

[8] Helaas gaat deze truc bij Borland niet op, vaak is de enig oplossing dan om dit met een assmbler progamma ook te doen.

filemappings in staan. De phantom file heet *$file.org*, het $teken zorgt er voor dat er geen name clashes onstaan aangezien we al hadden bepaald dat een $ teken in filenamen van het ext2fs worden geconverteerd naar een _ (underscore), zie voor een voorbeeld van $file.org het volgende hoodstuk waar een paar korte sessies worden beschreven.

redir.c:
Tot slot de redirection module, dit is tevens de *top-level* module. Naast de int2F interrupt handler, staat hier ook *main* (waar alle commanf line parameters worden geinterpreteerd). De redirector communiceert met niet direcet met het ext2fs, maar indirect via de vfs laag.
Hieronder staat een overzicht wat we met de redirection requests doen, de onderstaande lijst is een opsomming van alle (bekende) redirection calls op files.

- **change directory**, **close file**, **read file**, **set file attributes**, **open file**, **find first**, **find next**, **seek from end**, **diskspace**: ondersteund en geimplementeerd.
- **remove directory**, **make directory**, **commit file**, **write file**, **get file attributes**, **rename file**, **delete file**, **create file** : we zijn readonly, dus geven we in al deze gevallen *access_denied* terug.
- **inquiry**: Een redirector kan hiermee aangeven of het wenselijk is dat andere redirectors na hun geladen mogen worden. Van een redirector die zichzelf in de int2F keten wil hangen wordt verwacht dat hij eerst een *inquiry* uitvoert en zich schikt naar het antwoord (niet mogen is is dit geval niet hetzelfde als niet kunnen). Onze redirector wordt niet geladen na een negatief antwoord op de *inquiry*, na het laden geven we zelf een positief antwoord op binnengekomen inquiry requests.
- **shutdown**: dit is een zelf verzonnen request (van een ongebruikte redirection call) en wordt aangeroepen door de unload procedure. Het geeft de tsr de mogelijkheid om netjes af te sluiten (cache vrijgeven etc).
- **lock file**, **unlock file**: Deze worden niet ondersteund.
- **special open file**: dit wordt gebruikt om een file te openen. Indien de file bestaat kan worden aangegeven of deze gopend moet worden om te lezen of om hem te overschrijven. Indien de file bestaat kan worden aangegeven of deze gecreerd moet worden of niet. Alle pogingen om een file te schrijven (creeeren of overschrijven) resulteren in een *acces_denied* error, alle andere pogingen worden geaccepteerd.

7. Het Gebruik van MOUNTE2

In dit hoofdstuk wordt kort uitgelegd hoe MOUNTE2 gebruikt kan worden en wat de mogelijkheden van de command line parameters zijn. We zullen beginnen met de *usage message*[9]:

```
Usage:   mounte2 [hda3] d:

   hda3: linux partition on disk, first found if ommited
   d:  : drive on which the ext2fs should be mapped
Options:
   DC  : Debug to console (this is the default)
   DEx : Debug to ems, use x pages (defaults to 1)
   K   : Keep emspages after unloading
   Lx  : Debug level x (defaults to 0)
   M   : Maximum cache (512kb - 1mb)
   Nx  : # of char's to make a file unique (defaults to 1)
   SP  : Show partition tables
   SF  : Show list of debug flags
   U   : Unload latest version
```

In de meest eenvoudige vorm wordt MOUNTE2 als volgt gebruikt:

```
mounte2 z:
```

Hiermee wordt aangegeven dat de eerste ext2fs partitie die gevonden wordt, gemapt wordt op schijf z:. Alleen de schijfnaam is een verplichtte parameter. Uiteraard kan de partitie ook gespecificeerd worden, bv.:

```
mounte2 hdb2 z:
```

om een indicatie te geven van wat te verwachten laten we hier een screendump zien van het commado:

```
dir z:\usr\src
```

```
 Volume in drive Z has no label
 Directory of Z:\USR\SRC

.            <DIR>     06-12-95   2:09p
..           <DIR>     06-12-95   2:09p
PERL-4   032 <DIR>     05-15-95   9:32p
RCS          <DIR>     05-15-95   9:32p
NCURSES- 8_4 <DIR>     05-15-95   9:32p
LINUX                 11 05-03-95   8:28p
VGASET       <DIR>     05-15-95   9:32p
KERNEL-1 1_7      35830 02-09-95   9:20p
SENDMAIL     <DIR>     05-15-95   9:32p
PPP-2    1_9 <DIR>     05-15-95   9:32p
LINUX-1  2_0 <DIR>     06-12-95   2:09p
$FILES   ORG          0 06-12-95   2:09p
       12 file(s)      35841 bytes
                      7203840 bytes free
```

Wat hierin opvalt is de file `$FILES.ORG` , welke voorkomt in elke directory. Deze file laat de mapping zien van de geconverteerde filenamen en de originele filenamen in die directory. De inhoud van `$FILES.ORG` in \urs\src is hieronder afgebeeld.

[9] De source en alle meldingen in MOUNTE2 zijn in het Engels.

```
dwrxw-xw-x .            = .
dwrxw-xw-x ..           = ..
dwrxw-xw-x PERL-4   032 = perl-4.036
dwrxw-xw-x RCS          = rcs
dwrxw-xw-x NCURSES- 8_4 = ncurses-1.8.5
lwrxwrxwrx LINUX        = linux
dwrxw-xw-x VGASET       = vgaset
 wr-w--w-- KERNEL-1 1_7 = kernel-1.1.47+_patch
dwrxw-xw-x SENDMAIL     = sendmail
dwrxw-xw-x PPP-2    1_9 = ppp-2.1.2b
dwrxw-xw-x LINUX-1  2_0 = linux-1.2.3
```

We zullen nu de command line parameters van iets meer commentaar voozien:

- DC : Debug to Console. Hiermee worden alle debug messages naar het beeldscherm geschreven. Dit is eigenlijk alleen zinvol als schrijven naar expanded memory niet kan (omdat door een bug het systeem gereboot wordt, of iets in die geest)
- DEx : Debug to Ems, use x pages (defaults to 1). Debug messages worden naar expanded memory geschreven. Met een utitily als *ems.com* van Quarterdeck kunnen deze pagina's naar file worden geschreven. De handle die bij deze debug pages hoort is van de naam MOUNTE2 voorzien, dit vergemakkelijkt het zoeken naar de goede handle.
- K : Keep emspages after unloading. Deze optie is alleen zinvol in combinatie met -U. Op deze manier kan tijdens het unloaden worden aangegven of de ems pages vrijgevevn moeten worden of niet.
- Lx : debug Level x (defaults to 0). Het debuglevel specificeerd van welke debug klassen er messages moet worden geschreven. Welke vlaggen er zijn kan met optie /SF (show flag) worden bekeken.
- M : Maximum cache (512kb - 1mb). Gebruik maximum cache. Om 1 inode of diskblock te cachen is, behalve expanded memory, ook conventioneel geheugen nodig (22 bytes per item). De grootte van de maximum cache wordt dan ook niet bepaald door het vrije expanded memory maar door het nog vrij geheugen van de TSR binnen het 64 KB segement[10].
- Nx : Number of char's to make a file unique (defaults to 1). Aantal karkaters (1-3) dat gebruikt wordt om een filenaam uniek te maken.
- SP : Show Partition tables. Deze optie laat alleen de aanwezige partities zien en zal geen redirector laden. Overigens kan met opties -l64 de partities ook worden bekeken tijdens het laden van de redirector.
- SF : Show list of debug Flags. Laat alle geldige vlaggen van de verschillende debugklassen zien.
- U : Unload latest version. Verwijdert de laatst instantie van MOUNTE2.

Tot slot bekijken hoe de partities afgedrukt worden met:

```
MOUNTE2 /sp
```

[10] De TSR snoept zelf zo'n 35-40 kb op, de rest is voor de cache datastructuur.

```
MOUNTE2 v.1.0
Part       1:hda1,   60 MB, DOS 16-bit >=32 (6)
Part       2:hda2,   40 MB, Extended (5)
Part       3:    hda5,   40 MB, DOS 16-bit >=32 (6)
Part       4:hda3,    1 MB, OS/2 Boot Manag (10)
Part       5:hdb1,   15 MB, Linux swap (130)
Part       6:hdb2,  109 MB, Linux native (131)
Part       7:hdc1,  200 MB, Linux native (131)
Part       8:hdc2,  200 MB, Extended (5)
Part       9:    hdc5,  100 MB, DOS 16-bit >=32 (6)
Part      10:    hdc6,  100 MB, DOS 16-bit >=32 (6)
Part      11:hdc3,  102 MB, DOS 16-bit >=32 (6)
```

De regels die nu op het scherm komen zijn in feite debugmessages van debugklasse *partition* Het inspringen van partities hda5, hdc5 en hdc6 geeft aan dat het een logische partitie binnen een extended partitie betreft.

8. Conclusies

In de vorige hoofdstukken hebben we MOUNTE2, een network redirector voor Dos, gepresenteerd. MOUNTE2 is een dynamic (un)loadable TSR die het second extended filesysteem (ext2fs) van Linux transparant onder Dos zichtbaar maakt. Op dit moment is MOUNTE2 alleen in staat om het ext2fs te lezen en dus niet te schrijven.

Onder de voordelen en features van MOUNTE2 kunnen worden gerekend:

- (un)loadable TSR: MOUNTE2 hoeft dus alleen geladen te zijn wanneer het nodig is.
- Cache: Diskblocken en inodes worden gecachet in expanded memory. MOUNTE2 neemt dan in totaal ten hoogste 64KB van conventional memory in beslag (zonder cache iets van 35 KB).
- Filenamen: MOUNTE2 laat in elke directory in de file $file.org de orignale filenamen zien.
- Debug: MOUNTE2 kan debug informatie printen naar zowel het beeldscherm als naar expanded memory (gestuurd met allerlei opties), dit is voor het debuggen van TSR's bijna onmisbaar.
- Structuur: Door de modulaire strcutuur is MOUNTE2 makkelijk uit te breiden; bovendien zijn componenten eenvoudig te hergebruiken in andere projecten.
- Vfs; MOUNTE2 is ontworpen met de een virtual filesysteem laag. Op deze manier kunnen andere filesystemen makkelijk worden geintegreerd binnen MOUNTE2.

Uiteraard zijn er ook nadelen en minder mooie kanten aan MOUNTE2:

- MOUNTE2 kan alleen lezen en niet schrijven.
- MOUNTE2 is gebaseerd op ongedocumenteerde features van Dos.
- MOUNTE2 leest en schrijft direct in Dos datastructuren (dit wordt weliswaar implicitiet toegestaan door Dos zelf).
- De Borland compiler kan niet gebruikt worden om MOUNTE2 te compileren.
- Vanwege implementatie technische redenen kan MOUNTE2 niet draaien zonder een expanded memory manager (versie 4.0).

MOUNTE2 draait onder zowel onder Dos (vanaf 3.0) als in Dos sessies onder Ms-Windows (enhanced mode). MOUNTE2 draait niet onder OS/2 omdat die, onder andere, de redirector interface niet ondersteund.Verder worden ext2 filesystemen met alle blocksizes ondersteund (1KB, 2KB en 4KB).

Literatuur Overzicht

[1] R. Brown, *Interrupt List*, release 42, inter42.zip, ftp-site: *simtel*.

[2] R. Duncan, C. Petzold, M. Baker, A. Schulman, S. Davis, R. Nelson en R. Moote, *Dos Uitbreidingen*, Addison-Wesley, 1990.

[3] R. Duncan, *Ms-Dos voor gevorderden*, 2de editie, Microsoft Press, 1990.

[4] R. Duncan, *Toegevoegde Ms-Dos Functies*, Kluwer, 1990.

[5] S. Leffler, M. McKusick, M. Karels en J. Quarterman, *The Design and Implementation of the BSD4.3* UNIX *Operating System*, Addison-Wesley, 1989.

[6] A. Schulman, R. Brown, D. Maxey. R. Michels en J. Kyle, *Undocumented Dos, a Programmers Guide to reserved Ms-Dos Functions and Data Structures*, 2de editie, Addison-Wesley, 1993.

[7] D. Williams, *Programmer's Technical Reference for MSDOS and the IBM PC*, shareware versie, dosref22.zip, ftp-site: *simtel.*

[8] A. Wyatt, Sr., *Advanced Assembly Language*, Que Corp., 1992.

Appendix A Sources

Sourcefile: Biosdisk.h

```
#ifndef __BIOSDISK_H
#define __BIOSDISK_H

#include "systypes.h"

int    bios_rsect(uint8 disk, uint8 nsect, uint16 cyl, uint8 sect, uint8 head, void far
*data);
int    bios_wsect(uint8 disk, uint8 nsect, uint16 cyl, uint8 sect, uint8 head, void far
*data);
voidbios_driveparam(uint8 disk, uint32* cyl, uint32* sec, uint32* head);

#endif    // __BIOSDISK_H
```

Sourcefile: Biosdisk.c

```
#include <dos.h>

#include "systypes.h"
#include "biosdisk.h"
#include "debug.h"

union    REGS    regsin,regsout;
struct   SREGS   regsseg;

#ifdef __BORLANDC__
#pragma warn -par
#endif

/*
 * returns 0 on error, number of sectors on succes
 */
int bios_rsect(uint8 disk, uint8 nsect, uint16 cyl, uint8 sect, uint8 head, void far *data)
{
                /*
                 * the 2 highest bits of a 10 bits cylinder are placed in
                 * the highest 2 bits of CL
                 */
    dprintf(D_BIOS,"bios_rsect, disk:%u, nsect%u, cyl:%u, sect:%u, head%u¥n",
       (unsigned) disk,(unsigned) nsect, cyl, (unsigned) sect, (unsigned) head);
        regsin.h.cl=(uint8) ((sect & 0x3F) | ((cyl >> 2) & 0xC0));
        regsin.h.ch=(uint8) (cyl & 0xFF);
```

```
        regsin.h.ah=0x02;
        regsin.h.al=nsect;
        regsin.h.dh=head;
        regsin.h.dl=disk+0x80;
        regsin.x.bx=FP_OFF( (char  far *) data);
        regsseg.es=FP_SEG( (char far *) data);

        int86x(0x13,&regsin,&regsout,&regsseg);

    dprintf(D_BIOS,"bios_rsect:%s, disk:%u, nsect%u, cyl:%u, sect:%u, head%u¥n",
       regsout.x.cflag?"FAILED":"succes",
       (unsigned) disk,(unsigned) nsect, cyl, (unsigned) sect, (unsigned) head);
    return !regsout.x.cflag;
}


int bios_wsect(uint8 disk, uint8 nsect, uint16 cyl, uint8 sect, uint8 head, void far *data)
{
        return 0;
}


void bios_driveparam(uint8 disk, uint32* cyl, uint32* sec, uint32* head)
{
    regsin.h.ah = 8;
    regsin.h.dl = disk+0x80;
    int86(0x13, &regsin, &regsout);
    if (regsout.x.cflag)
    {
         *sec    = (uint32)  0;
         *cyl    = (uint32) 0;
         *head   = (uint32)  0;
    }
    else
    {
         *sec    = (uint32) (regsout.h.cl & 0x3F) ;
         *cyl    = (uint32) (((uint16) regsout.h.ch) | (((uint16) (regsout.h.cl & 0xC0))
<< 2));
         *head   = (uint32) regsout.h.dh + 1 ;
    }
    if (*cyl > (uint32) 1024)
         *cyl= (uint32) 1024;

    dprintf(D_BIOS,"driveparam, disk %u, nr_sec=%u, nr_cyl=%u, nr_head=%u¥n",
       (unsigned) disk, (unsigned) *sec, (unsigned) *cyl, (unsigned) *head);


}
#ifdef __BORLANDC__
#pragma warn +par
```

```
#endif
```

Sourcefile: Debug.h

```
#ifndef __DEBUG_H
#define __DEBUG_H

#define DOUT_EMS           1     /* dumps to ems           */
#define DOUT_CONSOLE     2     /* dumps to console         */
#define DOUT_NO_EOL      4     /* don't appends an eol   */

/*
 * If you add one, add a string in debug_prefix as well
 */

#define D_ALWAYS           0
#define D_SYSTEM              1
#define D_EMS                 2
#define D_BIOS                4
#define D_CACHE            8
#define D_DISKDEV            16
#define D_EXT2               32
#define D_PARTITION          64
#define D_REDIR          128
#define D_VFS                256
#define D_END                256
#define D_ALL               (D_END *2 -1)

void debug_init(int level, int out, unsigned ems_pages);
void debug_shutdown();
void debug_showlist();
int    debug_setlevel(int new_level);
int    debug_setout(int output);
void dprintf(int,char *format,...);

char press_key(char *message);


#endif
```

Sourcefile: Debug.c

```
#pragma check_stack(off)
#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
```

```
#include "debug.h"
#include "systypes.h"
#include "ems.h"
#include "vsprintf.h"

#define   DEBUG_STR_MAX 128   /* keep it a power of 2 */


static char           emsDebugName[]="MOUNTE2";
static union REGS     regs;
static emshandle_t    emsDebug=0;      /* debug to ems        */
static unsigned          emsPages;           /* number of ems pages */
static unsigned long  ems_pointer=0;   /* ems "filepointer"      */

static int            debug_level=0;
static int            debug_out=DOUT_CONSOLE;
static char        str_debug[DEBUG_STR_MAX];
static char        *debug_prefix[]= {"", "System", "Ems", "Bios", "Cache", "Disk", "Ext2",
"Part", "Redir", "Vfs"} ;


/*
 * fill ems debug pages with eol
 */
static void debug_clearems()
{
 unsigned page,offset;

 memset(str_debug,'¥n',DEBUG_STR_MAX);
 for (page=0 ; page< emsPages ; page++)
    for (offset=0 ; offset < PAGE_SIZE ; offset+=DEBUG_STR_MAX)
       emsConv2Ems(str_debug, emsDebug, page,offset, DEBUG_STR_MAX);
}

int debug_setlevel(int newlevel)
{
 int old=debug_level;

 debug_level=newlevel;

 return old;
}

int debug_setout(int newout)
{
 int old=debug_out;

 debug_out=newout;

 return old;
```

```
}

/*
 * Initialize debug module. Set debuglevel, outputmedium (ems/console) and
 * the number of ems pages (if output to ems).
 * We always bebug to ems with a handle called 'MOUNTE2', so we first
 * see if such a handle already exists, if so, we use that one (after
 * resizing the number of pages), otherwise we allocate new pages and the
 * according handle to 'MOUNTE2'
 */

void debug_init(int level,int out, unsigned ems_pages)
{
 debug_level=level;
 debug_out=out;
 emsPages=ems_pages;

 if (debug_out & DOUT_EMS)
 {
    if (!(emsDebug=emsGetHandle(emsDebugName)))
       if (! (emsDebug=emsAllocPages(emsPages)))
       {
          debug_setout(DOUT_CONSOLE);
          dprintf(D_ALWAYS,"Could not alloc ems pages for debugging¥n");
          return;
       }
       else
          emsSetHandleName(emsDebug,emsDebugName);
    else
    {
       emsReAllocPages(emsDebug,emsPages);
       //emsPages=emsNumPages(emsDebug);
    }
    debug_clearems();
    dprintf(D_SYSTEM,"Debug: handle:%u, Pages:%u¥n",emsDebug,emsPages);
 }
}

void debug_shutdown()
{
 uint16 debugHandle;

/*
 * We should only clean up a handle with our debugname, this way
 * the unload routine can decide whethet or not to keep the debug
 * pages.
 */

 if ((debugHandle=emsGetHandle(emsDebugName)))
    emsFreePages(debugHandle);
```

```
}

/*
 * dprintf writes a debug message to the current outputmedium
 * (ems/console).
 * First we check the flag against the current debuglevel to see if this
 * message should be printed in the first place. Every debugclass has it's
 * own prefix, except for D_ALWAYS which has none.
 */

void dprintf(int flag,char *format,...)
{
 va_list    argptr;
 int           which;
 unsigned      len;
 static int    debug_count=1;


 if ( !(((debug_level & flag) || !flag) && flag<=D_END))
    return;

 va_start(argptr, format);

 if (!flag)
    which=0;
 else
 {
    which=1;
    while (!(flag & 1))
    {
       which++;
       flag>>=1;
    }
 }

 len=0;
 if (flag)
    len=simple_sprintf(str_debug,"%-6s %5u:", debug_prefix[which], debug_count++);

 len+=simple_vsprintf(str_debug+len,format,argptr);
 va_end(argptr);
 if (!(debug_out & DOUT_NO_EOL))
 {
    strcat(str_debug,"¥r");
    len++;
 }

 if (debug_out & DOUT_CONSOLE)
 {
    strcat(str_debug,"$");
```

```
    regs.x.dx=(unsigned) str_debug;
    regs.h.ah=0x09;
    int86(0x21,&regs,&regs);
 }
 else
 {
    len--;
    if (ems_pointer + len > (unsigned long) emsPages*(unsigned long) PAGE_SIZE)
       ems_pointer=0l;
    emsConv2Ems((void far *) str_debug, emsDebug, (unsigned) (ems_pointer / PAGE_SIZE),
(unsigned) (ems_pointer % PAGE_SIZE), len);
    ems_pointer+=len;
 }
}


/*
 * Print the current list of debugflags known to the system
 */

void debug_showlist()
{
 int i,flag;
 for (i=flag=1 ; flag<=D_END ; i++)
 {
    dprintf(D_ALWAYS,"Flag %4u = %s¥n",flag,debug_prefix[i]);
    flag<<=1;
 }
}


/*
 * This dumps a message to the console and waits for a key.
 * Useful if a bug causes a reboot or something.
 */

char press_key(char *message)
{
 int     outold;

 if (message)
 {
    outold=debug_setout(DOUT_CONSOLE);
    dprintf(D_ALWAYS,message);
    debug_setout(outold);
 }

 regs.h.ah=0;
 int86(0x16,&regs,&regs);
 return regs.h.al;
```

```
}
```

Sourcefile: Diskdev.h

```
#ifndef __DISKDEV_H
#define __DISKDEV_H


#include "systypes.h"

#define BYTES_PERSECTOR 512
#define MAX_HD 4
#define MAX_LDEV 16
#define TYPE_EXT_PARTITION 5

#define BLOCKSIZE(d)   (d->sec_perblock*512)

typedef struct
{
   uint8      boot_ind;     /* 0x80 - active                    */
   uint8      head;         /* starting head                    */
   uint8      sector;       /* starting sector                  */
   uint8      cyl;          /* starting cylinder                */
   uint8      sys_ind;   /* What partition type              */
   uint8      end_head;     /* end head                              */
   uint8      end_sector;/* end sector                       */
   uint8      end_cyl;   /* end cylinder                     */
   uint32     start_sec;    /* starting sector counting from 0   */
   uint32     nr_sec;       /* nr of sectors in partition        */
} partition_t;


typedef struct
{
   uint32     maxhead;
   uint32     maxcyl;
   uint32     maxsec;
} pdev_t;

int   pdev_read(uint8 disk, uint32 start_sec, uint8 nr_sec, void far *data);
int   pdev_write(uint8 disk, uint32 start_sec, uint8 nr_sec, void far * data);

typedef struct
{
   int32   start_sec;
   int32   nr_sec;

   uint8   disk;           /* disk number             */
```

```
   uint8    type;            /* match sys_ind above      */
   uint8    partition;       /* partion number           */
   uint8    sec_perblock; /* # sectors per block      */
   void     *fs_info;        /* file system depend info  */
} ldev_t;

int      ldev_read(ldev_t *ldev, uint32 start_sec, uint8 nr_sec, void far *data);
int      ldev_write(ldev_t *ldev, uint32 start_sec, uint8 nr_sec, void far *data);
int      ldev_readblock(ldev_t *ldev, uint32 block, void far *data);
int      ldev_writeblock(ldev_t *ldev, uint32 block, void far *data);
int      ldev_setblocksize(ldev_t *ldev, uint32 bsize);
void  pdev_readpartitions();


#ifdef DECL_DEVICES
pdev_t pDevices[MAX_HD];
ldev_t lDevices[MAX_HD][MAX_LDEV];
#else
extern pdev_t pDevices[MAX_HD];
extern ldev_t lDevices[MAX_HD][MAX_LDEV];
#endif
#undef DECL_DEVICES


#endif     // __DISKDEV_H
```

Sourcefile: Diskdev.c

```
#pragma check_stack(off)
#include <stdlib.h>
#ifdef __BORLANDC__
#include <mem.h>
#else
#include <memory.h>
#endif

#include "systypes.h"
#define  DECL_DEVICES
#include "diskdev.h"
#include "biosdisk.h"
#include "debug.h"

/*
 * This module implements the concept of devices. It reads the partition
 * tables and creates an global array for all devices; this list
 * is extern'ed in the header file. Read and writesector routines may be
 * used for accessing an individual partition. By setting a blocksize, you
 * can also read and write blocks.
```

```
 */


int fs_type_list[]={1,4,5,6,7,0x0A,0x51,0x64,0x80,0x81,0x82,0x83,0x93,0x94,0};
char *fs_type_name[]={"DOS 12-bit FAT","DOS 16-bit <32M", "Extended",
                "DOS 16-bit >=32", "OS/2 HPFS","OS/2 Boot Manag",
                "Novell?","Novell", "Old MINIX","Linux/MINIX",
                "Linux swap","Linux native", "Amoeba","Amoeba BBT",
                "Unknown"};

static char *find_fstype(int type)
{
   int   i=0;
   for(i=0; fs_type_list[i] && fs_type_list[i]!=type ; i++)
      ;
   return fs_type_name[i];
}

/*
 * Create devices for each logical partition in an extended partition.
 * The logical partitions form a linked list, with each entry being
 * a partition table with two entries.  The first entry is the real data
 * partition (with a start relative to the partition table start). The
 * second is a pointer to the next logical partition (with a start
 * relative to the entire extended partition).
 */

static uint8 read_extended_partitions(uint8 disk,uint8 cur_par)
{
   partition_t   *p;
   uint32        first_sector, this_sector;
   char          *data;

   data=(char *)malloc(1024);

   first_sector = lDevices[disk][cur_par].start_sec;
   this_sector = first_sector;

   while (cur_par<MAX_LDEV && ldev_read(&lDevices[disk][cur_par],0, 1,data))
   {
      if (*(uint16 *) (data+510) != 0xAA55)
         break;         // could have been in the while condition,
                        // but I rather not depend on ldev_read being
                        // called first

        /*
         * Process the first entry, which should be the real
         * data partition.
         */
```

```
      p = (partition_t *) (0x1BE + data);
      if (p->sys_ind == TYPE_EXT_PARTITION || !p->nr_sec)
         break;  /* shouldn't happen */

         // overwrite the info, it was only needed to read the tables
      IDevices[disk][cur_par].start_sec = this_sector + p->start_sec;
      IDevices[disk][cur_par].nr_sec=p->nr_sec;
      IDevices[disk][cur_par].type=p->sys_ind;
      IDevices[disk][cur_par].disk=disk;
      IDevices[disk][cur_par].partition=cur_par;

      dprintf(D_PARTITION,"    hd%c%c, %4lu MB, %s (%u)¥n",
         (int) disk+'a',(int) cur_par+'0', p->nr_sec >> 11,
         find_fstype(p->sys_ind),(unsigned) p->sys_ind);

      cur_par++;
      p++;

         /*
          * Process the second entry, which should be a link
          * to the next logical partition.  Create a fake partition
          * for it just long enough to get the next partition
          * table. The fake partition will be reused for the real
          * data partition.
          */


      if (p->sys_ind != TYPE_EXT_PARTITION || !p->nr_sec)
         break;  /* no more logicals in this partition */

      IDevices[disk][cur_par].start_sec = this_sector + p->start_sec;
      IDevices[disk][cur_par].type=p->sys_ind;
      IDevices[disk][cur_par].disk=disk;
      IDevices[disk][cur_par].partition=cur_par;

      this_sector = first_sector + p->start_sec;
   }
   free(data);
   return cur_par;
}


/*
 * There are at most 4 primary partitions, they reside in the first
 * sector of a disk. If a primary partition's type is 6, it's an extended
 * partition. We print each partion with flag D_PARTITION, in this way
 * all partitions are printed when the debuglevel includes this bit.
 */

static void read_all_partitions(uint8 disk)
```

```
{
   int            i;
   uint8          extended=5;
   partition_t    *p;
   char           *data;

   data=(char *)malloc(1024);


   memset(lDevices[disk],0,sizeof(ldev_t)*MAX_LDEV);

   if (!pdev_read((uint8) disk,0, 1,data))
      return;


      // check for signature
   if (*(uint16 *) (data+510) == 0xAA55)
   {
      p = (partition_t *) (0x1BE + data);
      for (i=1 ; i<=4 ; i++,p++)
      {
         if (!p->nr_sec)
            continue;

         dprintf(D_PARTITION,"hd%c%c, %4lu MB, %s (%u)¥n",
            (int) disk+'a',(int) i+'0', p->nr_sec >> 11,
            find_fstype(p->sys_ind),(unsigned) p->sys_ind);

         lDevices[disk][i].start_sec=p->start_sec;
         lDevices[disk][i].nr_sec=p->nr_sec;
         lDevices[disk][i].type=p->sys_ind;
         lDevices[disk][i].disk=disk;
         lDevices[disk][i].partition=i;

         if (p->sys_ind == TYPE_EXT_PARTITION)
         {
            /*
             * Now handle the extended partitions. Make a new logical
             * sothat we can read the first partition table on it.
             */

            lDevices[disk][extended].start_sec=p->start_sec;
            lDevices[disk][extended].disk=disk;
            lDevices[disk][extended].partition=extended;

            extended=read_extended_partitions(disk,extended);
         }
      }
   }
   free(data);
```

```
}

void pdev_readpartitions(void)
{
   uint8            i;
   pdev_t           *p;

   for (i=0 ; i<MAX_HD ; i++)
   {
      p=&pDevices[i];
      bios_driveparam(i,&p->maxcyl,&p->maxsec,&p->maxhead);
      if (p->maxsec)
         read_all_partitions(i);
   }
}

int ldev_read(ldev_t *ldev, uint32 start_sec, uint8 nr_sec, void far *data)
{

   return pdev_read(ldev->disk,ldev->start_sec+start_sec,nr_sec,data);
}

int ldev_write(ldev_t *ldev,uint32 start_sec, uint8 nr_sec, void far * data)
{
   return pdev_write(ldev->disk,ldev->start_sec+start_sec,nr_sec,data);
}


int pdev_read(uint8 disk, uint32 start_sec, uint8 nr_sec, void far *data)
{
   uint32      sec,track,head,cyl;
   uint16      rval;

   sec   = start_sec % pDevices[disk].maxsec +1;
   track = start_sec / pDevices[disk].maxsec;
   head  = track % pDevices[disk].maxhead;
   cyl   = track / pDevices[disk].maxhead;

   rval=bios_rsect(disk, (uint8) nr_sec, (uint16) cyl, (uint8) sec, (uint8) head, data);
   return rval;

}

int pdev_write(uint8 disk, uint32 start_sec, uint8 nr_sec, void far * data)
{
   uint32      sec,track,head,cyl;

   sec   = start_sec % pDevices[disk].maxsec + 1;
   track = start_sec / pDevices[disk].maxsec;
   head  = track % pDevices[disk].maxhead;
```

```
   cyl   = track / pDevices[disk].maxhead;

   return bios_wsect(disk, (uint8) nr_sec, (uint16) cyl, (uint8) sec, (uint8) head, data);
}

int ldev_readblock(ldev_t *ldev, uint32 block, void far *data)
{
   dprintf(D_DISKDEV,"readblock, disk:%u, partition:%u, block:%lu¥n",
      (unsigned) ldev->disk, (unsigned) ldev->partition, block);
   return ldev_read(ldev,block*ldev->sec_perblock,ldev->sec_perblock, data);
}

int ldev_writeblock(ldev_t *ldev, uint32 block, void far *data)
{
   dprintf(D_DISKDEV,"writeblock, disk:%u, partition:%u, block:%lu¥n",
      (unsigned) ldev->disk, (unsigned) ldev->partition, block);
   return ldev_write(ldev,block*ldev->sec_perblock,ldev->sec_perblock, data);
}

int ldev_setblocksize(ldev_t *ldev, uint32 bsize)
{
   unsigned sec_perblock = (unsigned) (bsize / (uint32) BYTES_PERSECTOR);

   if ( (uint32) sec_perblock * (uint32) BYTES_PERSECTOR != bsize)
   {
      dprintf(D_ALWAYS,"Blocksize %lu not supported on hd%c%c¥n", bsize, (char) ldev-
>disk+'c',(char) ldev->partition);
      return 0;
   }
   ldev->sec_perblock=sec_perblock;
   dprintf(D_DISKDEV,"setblocksize, disk:%u, partition:%u, size:%lu¥n",
      (unsigned) ldev->disk, (unsigned) ldev->partition, bsize);
   return 1;
}
```

Sourcefile: Ems.h

```
#ifndef __EMS_H
#define __EMS_H


/*
 * This implements the basic ems stuff for our driver. Most of
 * it are 3.0 calls, only the copy functions are 4.0
 */

#define PAGE_SIZE 16384
```

```
typedef  unsigned int emshandle_t;

#pragma pack(1)
typedef  struct
{
   unsigned long  len;
   char              src_type;
   emshandle_t    src_handle;
   unsigned       src_offset;
   unsigned       src_seg_or_page;
   char              dst_type;
   emshandle_t    dst_handle;
   unsigned       dst_offset;
   unsigned       dst_seg_or_page;
} emscopy_t;
#pragma pack()

int      emsPresent();


/*
 * Info routines
 *
 */

unsigned    emsNumFreePages();
unsigned    emsNumTotPages();
unsigned emsNumPages(emshandle_t);
char*       emsGetVersion();


/*
 * Alloc/free
 *
 */

emshandle_t emsAllocPages(unsigned int Pages);
void           emsReAllocPages(emshandle_t Handle,unsigned int Pages);
void           emsFreePages(emshandle_t Handle);

/*
 *  Handle Names
 */

void emsSetHandleName(emshandle_t Handle, char * HandleName);
emshandle_t emsGetHandle(char * HandleName);

/*
 * Map a Page (pPage=[0..3] ; lPage=[0..n-1], n=number of alloced pages)
 *
```

```
 */

int      emsMapPage(emshandle_t Handle, char pPage, unsigned lPage);


/*
 * Save/restore context
 *
 */
void     emsSaveMap(emshandle_t);
void     emsRestoreMap(emshandle_t);

/*
 * Some copy routines, these are 4.0, they do not touch the context
 * mapping. Note the far pointers.
 */

void emsConv2Ems(void far *src, emshandle_t emsHandle, unsigned Page, unsigned offset,
unsigned long size);

void emsEms2Conv(emshandle_t emsHandle, unsigned Page, unsigned offset, void far
*dest,unsigned long size);

void emsEms2Ems(emshandle_t src_emsHandle, unsigned src_Page, unsigned src_offset,
emshandle_t dst_emsHandle, unsigned dst_Page, unsigned dst_offset, unsigned long size);


#ifdef EMS_DECL
char far * emsPageFrame;
#else
extern char far * emsPageFrame;
#endif
#undef EMS_DECL

#endif     // __EMS_H
```

Sourcefile: Ems.c

```
#pragma check_stack(off)
#include <dos.h>

#define EMS_DECL
#include "ems.h"
#include "debug.h"
#include "vsprintf.h"

#ifndef MK_FP
#define MK_FP(a,b)   ((void far *)(((unsigned long)(a) << 16) | (b)))
```

```
#endif



static char            emsDriverName[]="EMMXXXX0";
static char         emsVersionString[8]="0.0";
static char         emsMapBuffer[256];
static emscopy_t    copy_buffer;
static union REGS   emsregs;
static struct SREGS emssegregs;


static int          emsGetPageFrame();
static int             emsGetSizeMapInfo();


static void emsErrorMessage(char *func)
{
   dprintf(D_EMS,"ems error in:%s, error:%u¥n",func,(unsigned) emsregs.h.ah);
}

static void emsVersion()
{

   emsregs.h.ah=0x46;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsVersion");

   simple_sprintf(emsVersionString,"%u.%u",(unsigned) (emsregs.h.al & 0x00F0) >> 4,
(unsigned) emsregs.h.al & 0x000F);
}


static void emsCopy(emscopy_t *copybuf)
{
   emsregs.h.ah=0x57;
   emsregs.h.al=0x00;
   emsregs.x.si=(unsigned)copybuf;    /* note that copybuf must be a DS:SI pointer */
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsCopy");
}

int emsPresent()
{
   _asm {
      mov  al,67h
      mov  ah,35h
      int  21h
```

```
      mov  di,10
      mov  si,offset emsDriverName
      mov  cx,8
      cld
      repz cmpsb
      jnz  error

   }
   emsVersion();
   return emsGetPageFrame();

error:
#ifdef __BORLANDC__
   emsregs.h.ah=_AH;
#else
   _asm mov emsregs.h.ah,ah;
#endif
   emsErrorMessage("emsPresent");
   return 0;
}


unsigned int emsNumFreePages()
{
   emsregs.h.ah=0x42;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsNumFreePages");
   return emsregs.x.bx;
}

unsigned int emsNumTotPages()
{
   emsregs.h.ah=0x42;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsNumTotPages");
   return emsregs.x.dx;
}

unsigned emsNumPages(emshandle_t Handle)
{
   emsregs.h.ah=0x4C;
   emsregs.x.dx=Handle;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsNumPages");
   return emsregs.x.bx;
}
```

```
int emsGetPageFrame()
{

   emsregs.h.ah=0x41;
   int86(0x67,&emsregs,&emsregs);

   if (!emsregs.h.ah)
   {
      emsPageFrame=(char far *) MK_FP(emsregs.x.bx,0);
      return 1;
   }
   else
   {
      emsErrorMessage("emsGetPageFrame");
      emsPageFrame=(char far *) MK_FP(0,0);
      return 0;
   }
}


char *emsGetVersion()
{
   return emsVersionString;
}

emshandle_t emsAllocPages(unsigned int Pages)
{
   emsregs.h.ah=0x43;
   emsregs.x.bx=Pages;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
   {
      emsErrorMessage("emsAllocPages");
      return (emshandle_t) 0;
   }
   else
      return (emshandle_t) emsregs.x.dx;
}

void emsReAllocPages(emshandle_t Handle, unsigned Pages)
{
   emsregs.h.ah=0x51;
   emsregs.x.bx=Pages;
   emsregs.x.dx=Handle;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsReAllocPages");
```

```
}

void emsFreePages(emshandle_t Handle)
{
   emsregs.h.ah=0x45;
   emsregs.x.dx=Handle;
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsFreePages");

}

void emsSetHandleName(emshandle_t Handle, char * HandleName)
{
   emsregs.h.ah=0x53;
   emsregs.h.al=0x01;
   emsregs.x.si=(unsigned) HandleName;  /* note that HandleName must be a DS:SI pointer */
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsSetHandleName");
}

emshandle_t emsGetHandle(char * HandleName)
{
   emsregs.h.ah=0x54;
   emsregs.h.al=0x01;
   emsregs.x.si=(unsigned) HandleName;  /* note that HandleName must be a DS:SI pointer */
   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsSetHandle");

   return emsregs.x.dx;
}

int emsMapPage(emshandle_t Handle, char pPage, unsigned lPage)
{
   emsregs.h.ah=0x44;
   emsregs.h.al=pPage;
   emsregs.x.bx=lPage;
   emsregs.x.dx=Handle;

   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
   {
      emsErrorMessage("emsMapPage");
      return 0;
   }
   else
      return 1;
}
```

```
void emsSaveMap(emshandle_t Handle)
{
   dprintf(D_EMS,"SaveMap: %i bytes are needed¥n",emsGetSizeMapInfo());

   emsregs.h.ah=0x4e;
   emsregs.h.al=0x00;
   emsregs.x.di=(unsigned) emsMapBuffer;
#ifdef __BORLANDC__
   emssegregs.es=_DS;
#else
   _asm mov emssegregs.es, ds
#endif

   int86x(0x67,&emsregs,&emsregs,&emssegregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsSaveMap");
}
void emsRestoreMap(emshandle_t Handle)
{
   emsregs.h.ah=0x4e;
   emsregs.h.al=0x01;
   emsregs.x.si=(unsigned) emsMapBuffer;

   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsRestoreMap");
}

int emsGetSizeMapInfo()
{

   emsregs.h.ah=0x4E;
   emsregs.h.al=0x03;

   int86(0x67,&emsregs,&emsregs);
   if (emsregs.h.ah)
      emsErrorMessage("emsGetSizeMapMap");

   return (int) emsregs.h.al;
}


void emsConv2Ems(void far *src, emshandle_t emsHandle, unsigned Page, unsigned offset,
unsigned long size)
{
   unsigned long p_addr= ((((unsigned long) FP_SEG(src)) >> 4) << 8)
                    + (unsigned long )FP_OFF(src);

   copy_buffer.len=size;
```

```
   copy_buffer.src_type='¥0';
   copy_buffer.src_offset=FP_OFF(src);
   copy_buffer.src_seg_or_page=FP_SEG(src);

   copy_buffer.dst_type='¥1';
   copy_buffer.dst_handle=emsHandle;
   copy_buffer.dst_offset=offset;
   copy_buffer.dst_seg_or_page=Page;
   emsCopy(&copy_buffer);
}

void emsEms2Conv(emshandle_t emsHandle, unsigned Page, unsigned offset, void far
*dst,unsigned long size)
{
   copy_buffer.len=size;
   copy_buffer.src_type='¥1';
   copy_buffer.src_handle=emsHandle;
   copy_buffer.src_offset=offset;
   copy_buffer.src_seg_or_page=Page;
   copy_buffer.dst_type='¥0';
   copy_buffer.dst_offset=FP_OFF(dst);
   copy_buffer.dst_seg_or_page=FP_SEG(dst);
   emsCopy(&copy_buffer);

}

void emsEms2Ems(emshandle_t src_emsHandle, unsigned src_Page, unsigned src_offset,
emshandle_t dst_emsHandle, unsigned dst_Page, unsigned dst_offset, unsigned long size)
{
   copy_buffer.len=size;
   copy_buffer.src_type='¥1';
   copy_buffer.src_handle=src_emsHandle;
   copy_buffer.src_offset=src_offset;
   copy_buffer.src_seg_or_page=src_Page;
   copy_buffer.dst_type='¥1';
   copy_buffer.dst_handle=dst_emsHandle;
   copy_buffer.dst_offset=dst_offset;
   copy_buffer.dst_seg_or_page=dst_Page;
   emsCopy(&copy_buffer);

}
```

Sourcefile: Emscache.h

```
#ifndef __EMSCACHE_H
#define __EMSCACHE_H

#include "ems.h"
```

```
/*
 * Buffer chache is used for both blocks & inodes;
 * note that the cache resides completely in ems
 *
 */


typedef struct _cache_entry
{
   unsigned long       item_no;    /* nr of item were are caching */
   unsigned char       l_unit;       /* object's logical unit       */
   unsigned char       level;        /* level 1 or 2                   */
   unsigned            hashindex;
   unsigned            emsIndex;     /* where were put it in ems    */
   void far            *data;        /* if mapped in conv. mem      */
   struct _cache_entry *lru_next;    /* last recently use list      */
   struct _cache_entry *lru_prev;    /* last recently use list      */
   struct _cache_entry *hash_next; /* for speedy looking up       */
   struct _cache_entry *hash_prev; /* for speedy looking up       */
} cache_entry_t;


typedef struct
{
   cache_entry_t **hash_queue;
   unsigned   hash_size;
   emshandle_t   emsHandle;
   unsigned   nr_items;
   unsigned   size_item;
   cache_entry_t *lru_head[2];
   cache_entry_t *cache_list;
   unsigned long lookups;
   unsigned long hits;
} buffer_cache_t;

buffer_cache_t*     mk_cache(unsigned emsPages,unsigned size_item);
void                dl_cache(buffer_cache_t *bcache);
cache_entry_t*   cache_lookup(buffer_cache_t *bcache, unsigned l_unit, unsigned long
item_no);
cache_entry_t*   cache_add(buffer_cache_t *bcache, unsigned l_unit, unsigned long
item_no, void far *data);
void                cache_walk(buffer_cache_t *bcache, int level);

#endif     /* __EMSCACHE_H */
```

Sourcefile: Emscache.c

```
#pragma check_stack(off)
#include <stdlib.h>
#include <dos.h>

#include "debug.h"
#include "ems.h"
#include "emscache.h"
#include "tsr.h"

extern emshandle_t        emsCachePage;


/*
 * This module implements a cache buffer mechansism used
 * for inodes and diskblocks.
 * It is a two-level cache to prevent flushing when reading
 * "uninterresting" items. An item is lifted from level 1 to 2
 * after a cache-hit.
 *
 */

/* specify the number of items we idealy get on a queue */

#define HASH_QUEUE_SIZE 4
#define LEVEL1 '¥1'
#define LEVEL2 '¥2'

/* simple hashfun */
#define HASH_FUN(nr,sz)   ((unsigned) ( nr % (unsigned long) sz))



/*
 *  remove entry, special care if were are the lru_head
 */

static void lru_remove(buffer_cache_t *bcache, cache_entry_t *entry)
{
   cache_entry_t **lru= &(bcache->lru_head[entry->level-1]);

   dprintf(D_CACHE,"removing item:%lu, size:%u¥n",entry->item_no,bcache->size_item);
   if (entry==(*lru))
      (*lru)=(*lru)->lru_next;
   entry->lru_prev->lru_next=entry->lru_next;
   entry->lru_next->lru_prev=entry->lru_prev;
}



/*
 * insert entry e1 before e2
```

```
 */
static void lru_putbefore(cache_entry_t *e1,cache_entry_t *e2)
{
   e1->lru_next=e2;
   e1->lru_prev=e2->lru_prev;
   e2->lru_prev->lru_next=e1;
   e2->lru_prev=e1;
}



static void hash_remove(cache_entry_t **hashqueue, cache_entry_t* entry)
{
   if (entry->hash_next==entry)
         /* removal of last entry */
      hashqueue[entry->hashindex]=0;
   else
   {
      entry->hash_next->hash_prev=entry->hash_prev;
      entry->hash_prev->hash_next=entry->hash_next;
         /* ensure that that queue points to something */
      hashqueue[entry->hashindex]=entry->hash_next;
   }
}

static void hash_add(cache_entry_t **hashqueue, cache_entry_t* entry)
{
   cache_entry_t *hash_first=hashqueue[entry->hashindex];

   if (hash_first)
   {
      entry->hash_next=hash_first;
      entry->hash_prev=hash_first->hash_prev;
      hash_first->hash_prev->hash_next=entry;
      hash_first->hash_prev=entry;
   }
   else
   {
      entry->hash_next=entry;
      entry->hash_prev=entry;
   }

   hashqueue[entry->hashindex]=entry;
}



/*
 * We have an entry, put it last in the list (ie the most recent one)
 */
```

```
static void cache_update(buffer_cache_t *bcache,cache_entry_t *entry)
{

   lru_remove(bcache,entry);
   lru_putbefore(entry,bcache->lru_head[entry->level-1]);
}

/*
 * We replace 'entry' from level 1 with the lastest used entry from
 * level2 because we made 'entry' the most recent in level 1, the
 * level2_entry will now be the most recent entry in level 1 (seems
 * fair enough). Finally we mark 'entry' in level 2 as the most recent
 * one.
 */

static void cache_liftlevel(buffer_cache_t *bcache,cache_entry_t * entry)
{
   cache_entry_t *lru_head2=bcache->lru_head[1];
/*
 * first remove them
 */

   lru_remove(bcache,lru_head2);
   lru_remove(bcache,entry);

/*
 * switch level
 */

   lru_head2->level=LEVEL1;
   entry->level=LEVEL2;

/*
 * put them in the right position, that is most recently used
 */
   lru_putbefore(lru_head2,bcache->lru_head[0]);
   lru_putbefore(entry,bcache->lru_head[1]);
}

buffer_cache_t* mk_cache(unsigned emsPages,unsigned size_item)
{
   buffer_cache_t   *bcache;
   cache_entry_t    *entry;
   unsigned      i,halfway;


   if (! (bcache=(buffer_cache_t *)tsr_calloc(1,sizeof(buffer_cache_t))))
   {
      dprintf(D_ALWAYS,"Fatal could not alloc buffer_cache¥n");
```

```
      goto error_init_cache_1;
   }

   bcache->nr_items=(PAGE_SIZE/size_item)*emsPages;
   bcache->size_item=size_item;
   if ( !(bcache->emsHandle=emsAllocPages(emsPages)))
   {
      dprintf(D_ALWAYS,"Could not alloc %u emspages for cache entries¥n", emsPages);
      goto error_init_cache_2;
   }


   dprintf(D_CACHE,"Making cache with emshandle %u¥n",bcache->emsHandle);
   if ( !(bcache->cache_list=(cache_entry_t*) tsr_calloc(bcache-
>nr_items,sizeof(cache_entry_t))))
   {
      dprintf(D_ALWAYS,"Could not alloc cache entries¥n");
      goto error_init_cache_3;
   }
   dprintf(D_SYSTEM,"Alloced cache entries: %u¥n", bcache->nr_items * sizeof
(cache_entry_t));
   bcache->lru_head[0]=bcache->cache_list;
   halfway=bcache->nr_items/2;
   bcache->lru_head[1]=bcache->lru_head[0]+halfway;

/*
 * Init level 1 cache
 */
   i=0;
   entry=bcache->lru_head[0];
   entry->lru_prev=bcache->lru_head[1]-1;
   for (; i<halfway ; i++,entry++)
   {
      entry->hash_next=entry->hash_prev=entry;
      entry->lru_next=entry+1;
      entry->level=LEVEL1;
      entry->emsIndex=i;
      (entry+1)->lru_prev=entry;
   }
   (entry-1)->lru_next=bcache->lru_head[0];

/*
 * Init level 2 cache
 */

   entry=bcache->lru_head[1];
   entry->lru_prev=bcache->lru_head[1]+halfway-1;
   for( ; i<bcache->nr_items ; i++,entry++)
   {
      entry->hash_next=entry->hash_prev=entry;
```

```
      entry->lru_next=entry+1;
      entry->level=LEVEL2;
      entry->emsIndex=i;
      (entry+1)->lru_prev=entry;
   }
   (entry-1)->lru_next=bcache->lru_head[1];

/*
 * alloc hashqueue, initialised with 0 pointers
 */

   bcache->hash_size=bcache->nr_items/HASH_QUEUE_SIZE;
   if ( !(bcache->hash_queue=(cache_entry_t **)tsr_calloc(bcache->hash_size,
sizeof(cache_entry_t*))))
   {
      dprintf(D_ALWAYS,"Could not alloc hash_queue¥n");
      goto error_init_cache_4;
   }
   return bcache;

error_init_cache_4:
   free(bcache->cache_list);

error_init_cache_3:
   emsFreePages(bcache->emsHandle);

error_init_cache_2:
   free(bcache);

error_init_cache_1:
   return (buffer_cache_t*) 0;
}


/*
 * Delete a cache
 */


void dl_cache(buffer_cache_t *bcache)
{
   dprintf(D_SYSTEM,"Deleting cache with emshandle %u¥n",bcache->emsHandle);
   dprintf(D_SYSTEM,"lookups: %lu, hits:%lu¥n",bcache->lookups,bcache->hits);

   emsFreePages(bcache->emsHandle);
#ifndef RUNNING_AS_TSR
   free(bcache->cache_list);
   free(bcache->hash_queue);
   free(bcache);
#endif
```

```
}

/*
 * Lookup item.. returns 0 when not found
 */

cache_entry_t* cache_lookup(buffer_cache_t *bcache, unsigned l_unit, unsigned long
item_no)
{
   cache_entry_t    *entry,*hash_first;

   bcache->lookups++;

   hash_first=bcache->hash_queue[ HASH_FUN(item_no,bcache->hash_size) ];
   if (!hash_first)
      return (cache_entry_t *)0;

   entry=hash_first;
   do
   {
      if (item_no == entry->item_no && l_unit==(unsigned) entry->l_unit)
      {
         dprintf(D_CACHE,"lookup item (hit):%lu, size:%u¥n",item_no,bcache->size_item);
         bcache->hits++;
         if (entry->level==LEVEL1)
            cache_liftlevel(bcache,entry);
         else
            cache_update(bcache,entry);
         return entry;
      }
      else
         entry=entry->hash_next;

   } while (entry!=hash_first);
   dprintf(D_CACHE,"lookup item (miss):%lu, size:%u¥n",item_no,bcache->size_item);

   return (cache_entry_t *) 0;
}


/*
 * Add an item to the cache, it is made the most recent in level 1
 */

cache_entry_t* cache_add(buffer_cache_t *bcache, unsigned l_unit, unsigned long item_no,
void far *data)
{
   unsigned            newhashindex;
   unsigned            emsPage;
   unsigned long    offset;
```

```
   cache_entry_t*   entry;

   entry=bcache->lru_head[0];
   entry->item_no=item_no;
   entry->l_unit=(unsigned char) l_unit;

   offset=(unsigned long) bcache->size_item * (unsigned long) entry->emsIndex;
   emsPage=(unsigned) (offset / (unsigned long) PAGE_SIZE);
   offset=offset % (unsigned long) PAGE_SIZE;
// emsConv2Ems(data,bcache->emsHandle,emsPage,(unsigned) offset,bcache->size_item);
   emsEms2Ems(emsCachePage,0,FP_OFF(data)-0xC000,bcache->emsHandle,emsPage,(unsigned)
offset,bcache->size_item);
   entry->data=0;

   newhashindex=HASH_FUN(item_no,bcache->hash_size);
   if (newhashindex!=entry->hashindex)
   {
      hash_remove(bcache->hash_queue,entry);
      entry->hashindex=newhashindex;
      hash_add(bcache->hash_queue,entry);
   }
   bcache->lru_head[0]=entry->lru_next;

   dprintf(D_CACHE,"adding item%lu, size:%u¥n",item_no,bcache->size_item);
   return entry;
}

/*
 * This is useful when running under a debugger.
 */

void cache_walk(buffer_cache_t *bcache, int level)
{
   cache_entry_t    *first,*entry;

   entry=first=bcache->lru_head[level-1];
   do
   {
   } while ( (entry=entry->lru_next)!=first);
}
```

Sourcefile: Ext2_fs.h

```
#ifndef __EXT2_FS_H
#define __EXT2_FS_H

#include "systypes.h"
```

```
#include "emscache.h"
#include "diskdev.h"


/*
 * Most of these typedefs have shamelessly been copied from the
 * linux source
 */


#define EXT2_SUPER_MAGIC  0xEF53
#define EXT2_SIZE_SB         1024       /* size of superblock on disk */
/*
 * Constants relative to the data blocks
 */
#define  EXT2_NDIR_BLOCKS    12l
#define  EXT2_IND_BLOCK      EXT2_NDIR_BLOCKS
#define  EXT2_DIND_BLOCK        (EXT2_IND_BLOCK + 1l)
#define  EXT2_TIND_BLOCK        (EXT2_DIND_BLOCK + 1l)
#define  EXT2_N_BLOCKS       (EXT2_TIND_BLOCK + 1l)

/*
 * Inode flags
 */
#define  EXT2_SECRM_FL       0x00000001l /* Secure deletion */
#define  EXT2_UNRM_FL           0x00000002l /* Undelete */
#define  EXT2_COMPR_FL       0x00000004l /* Compress file */
#define EXT2_SYNC_FL           0x00000008l /* Synchronous updates */
#define EXT2_IMMUTABLE_FL    0x00000010l /* Immutable file */
#define EXT2_APPEND_FL       0x00000020l /* writes to file may only append */
#define EXT2_NODUMP_FL       0x00000040l /* do not dump file */


/*
 * Macro-instructions used to manage several block sizes
 */

#define EXT2_MIN_BLOCK_SIZE       1024l
#define  EXT2_MAX_BLOCK_SIZE      4096l
#define EXT2_MIN_BLOCK_LOG_SIZE   10l
#define EXT2_BLOCK_SIZE(s)         (EXT2_MIN_BLOCK_SIZE << (s) > s_log_block_size)
#define  EXT2_INODES_PER_BLOCK(s)  (EXT2_BLOCK_SIZE(s) / sizeof (ext2_inode))
#define EXT2_ACLE_PER_BLOCK(s)     (EXT2_BLOCK_SIZE(s) / sizeof (ext2_acl_entry))
#define  EXT2_ADDR_PER_BLOCK(s)    (EXT2_BLOCK_SIZE(s) / sizeof (uint32))

/*
 * Macro-instructions used to manage fragments
 */
#define EXT2_MIN_FRAG_SIZE        1024l
#define  EXT2_MAX_FRAG_SIZE       4096l
```

```
#define EXT2_MIN_FRAG_LOG_SIZE      10l


#define EXT2_FRAG_SIZE(s)          ( (s)->s_frag_size)
#define EXT2_FRAGS_PER_BLOCK(s) ( (s)->s_frags_per_block)


/*
 * Macro-instructions used to manage group descriptors
 */
#define EXT2_BLOCKS_PER_GROUP(s)   ( (s)->s_blocks_per_group)
#define EXT2_DESC_PER_BLOCK(s)     ( (s)->s_desc_per_block)
#define EXT2_INODES_PER_GROUP(s)   ( (s)->s_inodes_per_group)


/*
 * Structure of the super block on the disk
 */

typedef struct
{
   uint32   s_inodes_count;        /* Inodes count */
   uint32   s_blocks_count;        /* Blocks count */
   uint32   s_r_blocks_count;   /* Reserved blocks count */
   uint32   s_free_blocks_count;/* Free blocks count */
   uint32   s_free_inodes_count;/* Free inodes count */
   uint32   s_first_data_block; /* First Data Block */
   uint32   s_log_block_size;   /* Block size */
   int32       s_log_frag_size;    /* Fragment size */
   uint32   s_blocks_per_group; /* # Blocks per group */
   uint32   s_frags_per_group;  /* # Fragments per group */
   uint32   s_inodes_per_group; /* # Inodes per group */
   uint32   s_mtime;            /* Mount time */
   uint32   s_wtime;            /* Write time */
   uint16      s_mnt_count;           /* Mount count */
   int16       s_max_mnt_count;    /* Maximal mount count */
   uint16      s_magic;            /* Magic signature */
   uint16      s_state;            /* File system state */
   uint16      s_errors;              /* Behaviour when detecting errors */
   uint16      s_pad;
   uint32   s_lastcheck;           /* time of last check */
   uint32   s_checkinterval;    /* max. time between checks */
   uint32   s_creator_os;       /* OS */
   uint32   s_rev_level;           /* Revision level */
   uint32   s_reserved[236];    /* Padding to the end of the block */
} ext2_super_block;


/*
 * Special inodes numbers
```

```
 */
#define  EXT2_BAD_INO             1l    /* Bad blocks inode */
#define EXT2_ROOT_INO          2l    /* Root inode */
#define EXT2_ACL_IDX_INO       3l    /* ACL inode */
#define EXT2_ACL_DATA_INO      4l    /* ACL inode */
#define EXT2_BOOT_LOADER_INO   5l    /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO     6l    /* Undelete directory inode */
#define EXT2_FIRST_INO         11l   /* First non reserved inode */


/*
 * Structure of an inode on the disk
 * Please do not remove the union by removing the struct definiftions
 * for other OS's as it affects the size of struct ext2_inode
 */

typedef struct
{
   uint16     i_mode;          /* File mode */
   uint16     i_uid;           /* Owner Uid */
   uint32   i_size;          /* Size in bytes */
   uint32   i_atime;      /* Access time */
   uint32   i_ctime;      /* Creation time */
   uint32   i_mtime;      /* Modification time */
   uint32   i_dtime;      /* Deletion Time */
   uint16     i_gid;           /* Group Id */
   uint16     i_links_count;/* Links count */
   uint32   i_blocks;        /* Blocks count */
   uint32   i_flags;      /* File flags */
   union
   {
      struct
      {
         uint32  l_i_reserved1;
      } linux1;
      struct
      {
         uint32  h_i_translator;
      } hurd1;
      struct
      {
         uint32  m_i_reserved1;
      } masix1;
   } osd1;                                 /* OS dependent 1 */

   uint32   i_block[EXT2_N_BLOCKS];   /* Pointers to blocks */
   uint32   i_version;                   /* File version (for NFS) */
   uint32   i_file_acl;              /* File ACL */
   uint32   i_dir_acl;                   /* Directory ACL */
   uint32   i_faddr;                 /* Fragment address */
```

```
   union
   {
      struct
      {
         uint8  l_i_frag;    /* Fragment number */
         uint8  l_i_fsize;   /* Fragment size */
         uint16 i_pad1;
         uint32 l_i_reserved2[2];
      } linux2;
      struct
      {
         uint8  h_i_frag;    /* Fragment number */
         uint8  h_i_fsize;   /* Fragment size */
         uint16 h_i_mode_high;
         uint16 h_i_uid_high;
         uint16 h_i_gid_high;
         uint32 h_i_author;
      } hurd2;
      struct
      {
         uint8  m_i_frag;    /* Fragment number */
         uint8  m_i_fsize;   /* Fragment size */
         uint16 m_pad1;
         uint32 m_i_reserved2[2];
      } masix2;
   } osd2;                       /* OS dependent 2 */
} ext2_inode;

#define i_reserved1 osd1.linux1.l_i_reserved1
#define i_frag      osd2.linux2.l_i_frag
#define i_fsize        osd2.linux2.l_i_fsize
#define i_reserved2 osd2.linux2.l_i_reserved2


/*
 * Structure of a directory entry
 */
#define EXT2_NAME_LEN 255

typedef struct
{
   uint32  inode;              /* Inode number */
   uint16  rec_len;            /* Directory entry length */
   uint16  name_len;           /* Name length */
   char    name[EXT2_NAME_LEN]; /* File name */
} ext2_dir_entry ;



typedef struct
```

```
{
   uint32  bg_block_bitmap;        /* Blocks bitmap block */
   uint32  bg_inode_bitmap;        /* Inodes bitmap block */
   uint32  bg_inode_table;         /* Inodes table block */
   uint16  bg_free_blocks_count;   /* Free blocks count */
   uint16  bg_free_inodes_count;   /* Free inodes count */
   uint16  bg_used_dirs_count;     /* Directories count */
   uint16  bg_pad;
   uint32  bg_reserved[3];
} ext2_group_desc;



/*
 * ACL structures
 */
typedef struct
{
   uint32 aclh_size;
   uint32 aclh_file_count;
   uint32 aclh_acle_count;
   uint32 aclh_first_acle;
} ext2_acl_header;  /* Header of Access Control Lists */

typedef struct
{
   uint32 acle_size;
   uint16 acle_perms;  /* Access permissions */
   uint16 acle_type;   /* Type of entry */
   uint16 acle_tag;    /* User or group identity */
   uint16 acle_pad1;
   uint32 acle_next;   /* Pointer on next entry for the */
                       /* same inode or on next free entry */
} ext2_acl_entry;      /* Access Control List Entry */

/*
 * The following is not needed anymore since the descriptors buffer
 * heads are now dynamically allocated
 */
/* #define EXT2_MAX_GROUP_DESC  8 */

#define EXT2_MAX_GROUP_LOADED   8

/*
 * second extended-fs super-block data in memory
 */
typedef struct {
   uint32 s_frag_size;          /* Size of a fragment in bytes          */
   uint32 s_frags_per_block;    /* Number of fragments per block           */
   uint32 s_inodes_per_block;   /* Number of inodes per block              */
```

```
   uint32 s_frags_per_group;    /* Number of fragments in a group       */
   uint32 s_blocks_per_group;   /* Number of blocks in a group          */
   uint32 s_inodes_per_group;   /* Number of inodes in a group          */
   uint32 s_itb_per_group;      /* Num of inode table blocks per group     */
   uint32 s_db_per_group;       /* Number of descriptor blocks per group */
   uint32 s_desc_per_block;     /* Number of group descriptors per block   */
   uint32 s_groups_count;       /* Number of groups in the fs              */
   ext2_group_desc ** s_group_desc;
   uint16 s_loaded_inode_bitmaps;
   uint16 s_loaded_block_bitmaps;
   uint32 s_inode_bitmap_number[EXT2_MAX_GROUP_LOADED];
   char * s_inode_bitmap[EXT2_MAX_GROUP_LOADED];
   uint32 s_block_bitmap_number[EXT2_MAX_GROUP_LOADED];
   char * s_block_bitmap[EXT2_MAX_GROUP_LOADED];
} ext2_sb_info;


uint32           ext2_bmap(ldev_t *ldev,unsigned l_unit, ext2_inode far * inode, int32
block);
cache_entry_t*   ext2_readinode(ldev_t *ldev,unsigned l_unit, uint32 i_ino);
void             ext2_releaseinode(ldev_t *ldev, cache_entry_t *entry);
cache_entry_t*   ext2_readblock(ldev_t *ldev,unsigned l_unit, uint32 lblock);
void             ext2_releaseblock(ldev_t *ldev, cache_entry_t *entry);
int              ext2_init(ldev_t *ldev,unsigned ino_pages, unsigned block_pages);
void             ext2_shutdown(ldev_t *ldev);
void             ext2_savecontext(ldev_t *ldev);
void             ext2_restorecontext(ldev_t *ldev);
long             ext2_blockscount(ldev_t *ldev);
long             ext2_freeblockscount(ldev_t *ldev);

typedef struct
{
   ext2_super_block *ext2_sb;
   ext2_sb_info     *ext2_sbinfo;
} ext2_fs_info_t;

#endif     // __EXT2_FS_H
```

Sourcefile: Ext2_fs.c

```
#pragma check_stack(off)
#include <stdlib.h>
#include <string.h>
#include <dos.h>

#ifdef __BORLANDC__
#include <alloc.h>
#include <mem.h>
```

```
#else
#include <malloc.h>
#endif

#include "debug.h"
#include "diskdev.h"
#include "ext2_fs.h"
#include "emscache.h"
#include "tsr.h"


#define  CACHE_PAGE    3
emshandle_t         emsCachePage=0;
unsigned            MapSP=(unsigned) CACHE_PAGE* (unsigned) PAGE_SIZE;
char far*              CurCachePointer;

buffer_cache_t      *InodeCache;  /* Inode cache */
buffer_cache_t      *BlockCache;  /* Block cache */

/*
 * some neat shortcuts
 */
#define INODE_BMAP(inode,nr) ((inode)->i_block[(nr)])
#define BLOCK_BMAP(data,nr) (((unsigned long far *) data)[(nr)])
#define SUPERBLOCK(d)  (((ext2_fs_info_t *)d->fs_info)->ext2_sb)
#define SBINFO(d)   (((ext2_fs_info_t *)d->fs_info)->ext2_sbinfo)

/*
 * In order for other routines to read the contents of inodes and
 * diskblocks (which both reside in the emscache), we need to map
 * these items in conventional memory. For this we use the ems
 * PageFrame. Because we need some items at the same time, we use
 * a stack mechanism to push and pop items in the PageFrame. The
 * only thing these Push and Pop routines do is adjusting the Stack
 * Pointer.
 */

static char far* PushItem(unsigned size)
{
   char far *rval=CurCachePointer;

   if (MapSP + size < MapSP)
   {
           /* should not happen */
      press_key("PushItem wrapped around MapSP! Press a key to continue, but better is to
reboot¥n");
   }
   MapSP+=size;
   CurCachePointer+=size;
   return rval;
```

```
}


static void PopItem(unsigned size)
{
   if (MapSP - size < (unsigned) CACHE_PAGE* (unsigned) PAGE_SIZE)
   {
           /* should not happen */
      press_key("PopItem wrapped around MapSP! Press a key to continue, but better is to
reboot¥n");
   }
   MapSP-=size;
   CurCachePointer-=size;
}


static int ext2_read_sb(ldev_t *ldev)
{
   char far *          buffer;
   ext2_super_block *sb=SUPERBLOCK(ldev);
   ext2_sb_info        *sbinfo=SBINFO(ldev);
   uint32              logic_sb_block;
   uint32              desc_size;
   int                 db_count,i;

/*
 * The superblock is the logicalblock 1 on disk and has a size
 * of 1024 bytes
 */

   buffer=PushItem(EXT2_SIZE_SB);
   ldev_setblocksize(ldev,EXT2_SIZE_SB);
   if (!ldev_readblock(ldev,1,(void far *) sb))
   {
      dprintf(D_EXT2,"Could not read block for superblock¥n");
      PopItem(EXT2_SIZE_SB);
      return 0;
   }
   //_fmemcpy(sb, buffer, sizeof(ext2_super_block));
   PopItem(EXT2_SIZE_SB);

   if (sb->s_magic != EXT2_SUPER_MAGIC)
   {
      dprintf(D_EXT2,"Magic check failed¥n");
      return 0;
   }
/*
 * Set the blocksize for this device
 */
   if (!ldev_setblocksize(ldev,EXT2_BLOCK_SIZE(sb)))
```

```
   {
      dprintf(D_EXT2,"Could not set blocksize:%u¥n",(unsigned) EXT2_BLOCK_SIZE(sb));
      return 0;
   }

   logic_sb_block=sb->s_first_data_block;
   sbinfo->s_frag_size = EXT2_MIN_FRAG_SIZE <<  sb->s_log_frag_size;
   if (sbinfo->s_frag_size)
      sbinfo->s_frags_per_block=EXT2_BLOCK_SIZE(sb)/sbinfo-> s_frag_size;
   else
      sb->s_magic = 0;
   sbinfo->s_blocks_per_group = sb->s_blocks_per_group;
   sbinfo->s_frags_per_group = sb->s_frags_per_group;
   sbinfo->s_inodes_per_group = sb->s_inodes_per_group;
   sbinfo->s_inodes_per_block = EXT2_BLOCK_SIZE(sb) / sizeof (ext2_inode);
   sbinfo->s_itb_per_group = sbinfo->s_inodes_per_group / sbinfo->s_inodes_per_block;
   sbinfo->s_desc_per_block=EXT2_BLOCK_SIZE(sb)/sizeof (ext2_group_desc);

   sbinfo->s_groups_count = (sb->s_blocks_count -sb->s_first_data_block +
         EXT2_BLOCKS_PER_GROUP(sb) - 1) /  EXT2_BLOCKS_PER_GROUP(sb);
   db_count=(sbinfo->s_groups_count + EXT2_DESC_PER_BLOCK(sbinfo) - 1) /
         EXT2_DESC_PER_BLOCK(sbinfo);
   sbinfo->s_group_desc = (ext2_group_desc **) tsr_malloc (db_count * sizeof
(ext2_group_desc *));
   if (sbinfo->s_group_desc == NULL)
   {
      dprintf(D_EXT2,"EXT2-fs: not enough memory¥n");
      return 0;
   }
   for (i = 0; i < db_count; i++)
   {
      buffer=PushItem(BLOCKSIZE(ldev));
      if (!ldev_readblock(ldev,logic_sb_block + i + 1,buffer))
      {
         dprintf(D_EXT2,"Could not read block for group desc¥n");
         PopItem(BLOCKSIZE(ldev));
         return 0;
      }
      desc_size=sizeof(ext2_group_desc)*EXT2_DESC_PER_BLOCK(sbinfo);
      if ( !(sbinfo->s_group_desc[i]=(ext2_group_desc *) tsr_malloc ((size_t) desc_size)))
      {
         dprintf(D_EXT2,"Could not alloc group desc %i¥n",i);
         PopItem(BLOCKSIZE(ldev));
         return 0;
      }

      _fmemcpy(sbinfo->s_group_desc[i],buffer,(unsigned) desc_size);
      PopItem(BLOCKSIZE(ldev));
   }
/*
```

```
 * We should check the group descriptors here but we leave that
 * for the future :-)
 */
   for (i = 0; i < EXT2_MAX_GROUP_LOADED; i++)
   {
      sbinfo->s_inode_bitmap_number[i] = 0;
      sbinfo->s_inode_bitmap[i] = NULL;
      sbinfo->s_block_bitmap_number[i] = 0;
      sbinfo->s_block_bitmap[i] = NULL;
   }
   sbinfo->s_loaded_inode_bitmaps = 0;
   sbinfo->s_loaded_block_bitmaps = 0;
   sbinfo->s_db_per_group = db_count;

   return 1;
}


cache_entry_t* ext2_readinode(ldev_t *ldev, unsigned l_unit, uint32 i_ino)
{
   char far            *buffer;
   unsigned         emsPage;
   unsigned long    offset;
   ext2_inode       far *raw_inode;
   uint32              block_group,group_desc,desc, block;
   ext2_group_desc     *gdp;
   cache_entry_t    *entry;

   ext2_super_block *sb=SUPERBLOCK(ldev);
   ext2_sb_info        *sbinfo=SBINFO(ldev);

   if ((i_ino != EXT2_ROOT_INO && i_ino != EXT2_ACL_IDX_INO &&
       i_ino != EXT2_ACL_DATA_INO && i_ino < EXT2_FIRST_INO) ||
       i_ino > sb->s_inodes_count)
   {
      dprintf(D_EXT2,"ext2_read_inode,bad inode number: %lu¥n", i_ino);
      return 0;
   }

   buffer=PushItem(sizeof(ext2_inode));
   if ((entry=cache_lookup(InodeCache,l_unit,i_ino)))
   {
      offset= (unsigned long) InodeCache->size_item * entry->emsIndex;
      emsPage=(unsigned) (offset / (unsigned long) PAGE_SIZE);
      offset=offset % (unsigned long) PAGE_SIZE;
      emsEms2Ems(InodeCache->emsHandle,emsPage,(unsigned)
            offset,emsCachePage,0, FP_OFF(buffer)-(CACHE_PAGE * (unsigned)
PAGE_SIZE),InodeCache->size_item);
      entry->data=buffer;
```

```
      return entry;
   }

   block_group = (i_ino - 1) / EXT2_INODES_PER_GROUP(sbinfo);
   if (block_group >= sbinfo->s_groups_count)
   {
      dprintf(D_EXT2,"ext2_read_inode, group >= groups count¥n");
      PopItem(sizeof(ext2_inode));
      return 0;
   }

   group_desc = block_group / EXT2_DESC_PER_BLOCK(sbinfo);
   desc = block_group % EXT2_DESC_PER_BLOCK(sbinfo);
   gdp = sbinfo->s_group_desc[group_desc];
   block = gdp[desc].bg_inode_table +(((i_ino - 1) % EXT2_INODES_PER_GROUP(sb)) /
EXT2_INODES_PER_BLOCK(sb));

   entry=ext2_readblock(ldev,l_unit,block);
   raw_inode = ((ext2_inode far *) entry->data) + (i_ino - 1) % EXT2_INODES_PER_BLOCK(sb);
   _fmemcpy(buffer,raw_inode,sizeof(ext2_inode));
   ext2_releaseblock(ldev,entry);
   entry=cache_add(InodeCache,l_unit,i_ino,buffer);
   entry->data=buffer;

   return entry;
}


void ext2_releaseinode(ldev_t *ldev, cache_entry_t *entry)
{
   PopItem(sizeof(ext2_inode));
}


uint32 ext2_bmap(ldev_t *ldev,unsigned l_unit, ext2_inode far *inode, int32 block)
{
   cache_entry_t       *entry;
   uint32                 rval;
   ext2_super_block    *sb=SUPERBLOCK(ldev);
   int32                  addr_per_block = EXT2_ADDR_PER_BLOCK(sb);

   if (block < 0)
   {
      dprintf(D_EXT2,"ext2_bmap,block < 0¥n");
      return 0l;
   }

   if (block >= EXT2_NDIR_BLOCKS + addr_per_block +
          addr_per_block * addr_per_block +
          addr_per_block * addr_per_block * addr_per_block)
```

```
{
   dprintf(D_EXT2,"ext2_bmap, block > big¥n");
   return 0l;
}

if (block < EXT2_NDIR_BLOCKS)
   return (INODE_BMAP(inode, block));

block -= EXT2_NDIR_BLOCKS;
if (block < addr_per_block)
{
   rval = INODE_BMAP(inode, EXT2_IND_BLOCK);
   if (!rval)
      return 0l;

   entry=ext2_readblock(ldev,l_unit,rval);
   rval=BLOCK_BMAP(entry->data,block);
   ext2_releaseblock(ldev,entry);
   return rval;
}

block -= addr_per_block;
if (block < addr_per_block * addr_per_block)
{
   rval = INODE_BMAP(inode, EXT2_DIND_BLOCK);
   if (!rval)
      return 0l;
   entry=ext2_readblock(ldev,l_unit,rval);
   rval = BLOCK_BMAP(entry->data,block / addr_per_block);
   ext2_releaseblock(ldev,entry);
   if (!rval)
      return 0l;

   entry=ext2_readblock(ldev,l_unit,rval);
   rval=BLOCK_BMAP(entry->data,block & (addr_per_block - 1));
   ext2_releaseblock(ldev,entry);
   return rval;
}
block -= addr_per_block * addr_per_block;
rval = INODE_BMAP(inode, EXT2_TIND_BLOCK);
if (!rval)
   return 0l;

entry=ext2_readblock(ldev,l_unit,rval);
rval=BLOCK_BMAP(entry->data,block / (addr_per_block * addr_per_block));
ext2_releaseblock(ldev,entry);

if (!rval)
   return 0l;
```

```
   entry=ext2_readblock(ldev,l_unit,rval);
   rval=BLOCK_BMAP(entry->data,(block / addr_per_block) & (addr_per_block - 1));
   ext2_releaseblock(ldev,entry);
   if (!rval)
      return 0l;

   entry=ext2_readblock(ldev,l_unit,rval);
   rval=BLOCK_BMAP(entry->data,block & (addr_per_block - 1));
   ext2_releaseblock(ldev,entry);

   return rval;
}


cache_entry_t* ext2_readblock(ldev_t *ldev,unsigned l_unit, uint32 lblock)
{
   cache_entry_t *entry;
   char far *       buffer=PushItem(BLOCKSIZE(ldev));
   unsigned      emsPage;
   unsigned long offset;


   dprintf(D_EXT2,"ext2_readblock %lu¥n",lblock);
   if ((entry=cache_lookup(BlockCache,l_unit,lblock)))
   {
      offset= (unsigned long) BlockCache->size_item * entry->emsIndex;
      emsPage=(unsigned) (offset / (unsigned long) PAGE_SIZE);
      offset=offset % (unsigned long) PAGE_SIZE;

      emsEms2Ems(BlockCache->emsHandle,emsPage,(unsigned)
            offset,emsCachePage,0, FP_OFF(buffer)-(CACHE_PAGE * (unsigned)
PAGE_SIZE),BLOCKSIZE(ldev));
   }
   else
   {
      ldev_readblock(ldev,lblock,buffer);
      entry=cache_add(BlockCache,l_unit,lblock,buffer);
   }
   entry->data=(char far *) buffer;
   return entry;
}


void ext2_releaseblock(ldev_t *ldev, cache_entry_t *entry)
{
   PopItem(BLOCKSIZE(ldev));
}


/*
```

```
 * Oh boy, this is realy ugly...
 * Shall we restart the discusion on the use of gotot's ?
 *
 * The order in which we initialize is important. We start by
 * allocating the emsPageCache (which is used as a 64kb buffer),
 * allocate some memory (near heap!), read the superblock and
 * finaly allocate both inode and blockcache. In each routine
 * we can fail, in which case we must clean things up.
 */

int ext2_init(ldev_t *ldev, unsigned inodeCachePages, unsigned blockCachePages)
{

   if ( !(emsCachePage=emsAllocPages(1)))
   {
      dprintf(D_ALWAYS,"Could not alloc memory for emsCachePage¥n");
      goto error_ext2_init_1;
   }

   if (!emsMapPage(emsCachePage,CACHE_PAGE,0))
   {
      dprintf(D_ALWAYS,"Could not map cache page¥n");
      goto error_ext2_init_2;
   }
   CurCachePointer=emsPageFrame+(unsigned)CACHE_PAGE*(unsigned) PAGE_SIZE;

   if (!(ldev->fs_info=(ext2_fs_info_t*) tsr_calloc(1, sizeof( ext2_fs_info_t))))
   {
      dprintf(D_ALWAYS,"Could not alloc memory for ext2_file_info¥n");
      goto error_ext2_init_2;
   }

   if (!(SUPERBLOCK(ldev)=(ext2_super_block *) tsr_calloc(1, sizeof(ext2_super_block))))
   {
      dprintf(D_ALWAYS,"Could not alloc memory for ext2_super_block¥n");
      goto error_ext2_init_3;
   }

   if (!(SBINFO(ldev)=(ext2_sb_info *) tsr_calloc(1, sizeof(ext2_sb_info))))
   {
      dprintf(D_ALWAYS,"Could not alloc memory for sbinfo¥n");
      goto error_ext2_init_4;
   }

   if (!ext2_read_sb(ldev))
   {
      dprintf(D_ALWAYS,"Could not read super block¥n");
      goto error_ext2_init_5;
   }
```

```
   if (!(InodeCache=mk_cache(inodeCachePages,sizeof(ext2_inode))))
   {
      dprintf(D_ALWAYS,"Could not make inode cache¥n");
      goto error_ext2_init_5;
   }

   if (!(BlockCache=mk_cache(blockCachePages,BLOCKSIZE(ldev))))
   {
      dprintf(D_ALWAYS,"Could not make block cache¥n");
      goto error_ext2_init_6;
   }
   return 1;

error_ext2_init_6:
   dl_cache(InodeCache);

error_ext2_init_5:
   free(SBINFO(ldev));

error_ext2_init_4:
   free(SUPERBLOCK(ldev));

error_ext2_init_3:
   free(ldev->fs_info);

error_ext2_init_2:
   emsFreePages(emsCachePage);
   emsCachePage=0;


error_ext2_init_1:

   return 0;

}

/*
 * Free the caches, and the emsCachePage, and the malloc'ed memory.
 * NB restore the context first, otherwise emsCachePage wouldn't be freed
 */

void ext2_shutdown(ldev_t *ldev)
{
   dl_cache(BlockCache);
   dl_cache(InodeCache);
   emsRestoreMap(emsCachePage);
   emsFreePages(emsCachePage);
   emsCachePage=0;
#ifndef RUNNING_AS_TSR
   free(SBINFO(ldev));
```

```
   free(SUPERBLOCK(ldev));
   free(ldev->fs_info);
#endif
}


long ext2_blockscount(ldev_t *ldev)
{
   return SUPERBLOCK(ldev)->s_blocks_count;
}


long ext2_freeblockscount(ldev_t *ldev)
{
   return SUPERBLOCK(ldev)->s_free_blocks_count;
}


void ext2_savecontext(ldev_t *ldev)
{
   emsSaveMap(emsCachePage);
   emsMapPage(emsCachePage,CACHE_PAGE,0);
}


void ext2_restorecontext(ldev_t *ldev)
{
   if (emsCachePage)
      emsRestoreMap(emsCachePage);
}
```

Sourcefile: Redir.h

```
#ifndef __REDIR_H
#define __REDIR_H

#include "systypes.h"
#include "diskdev.h"

typedef void (far *FARPROC)(void);
typedef void (*PROC)(void);

#pragma pack(1)

/*
 * FindFirst/Next data block - ALL DOS VERSIONS
 */
typedef struct
```

```
{
        uint8           drive_no;
        char            srch_mask[11];
        uint8           attr_mask;
        uint16          dir_entry_no;
        uint16          dir_sector;
        uint8           f1[4];
} SRCHREC, far* SRCHREC_PTR;

/*
 * DOS System File Table entry - ALL DOS VERSIONS
 * Some of the fields below are defined by the redirector, and differ
 * from the SFT normally found under DOS
 */
typedef struct
{
        uint16          handle_count;
        uint16          open_mode;
        uint8           file_attr;
        uint16          dev_info_word;
        uint8 far *     dev_drvr_ptr;
        uint16          start_sector;
        uint32          file_time;
        int32           file_size;
        int32           file_pos;
        uint16          rel_sector;
        uint16          abs_sector;
        uint16          dir_sector;
        uint8           dir_entry_no;
        char            file_name[11];
} SFTREC, far* SFTREC_PTR;

/*
 * DOS Current directory structure - DOS VERSION 3.xx
 */
typedef struct
{
        char            current_path[67];
        uint16          flags;
        uint8           f1[10];
        uint16          root_ofs;
} V3_CDS, far* V3_CDS_PTR;


/*
 * DOS Current directory structure - DOS VERSION 4.xx
 */
typedef struct
{
        char            current_path[67];
```

```
        uint16          flags;
        uint8           f1[10];
        uint16          root_ofs;
        uint8           f2[7];
} V4_CDS, far* V4_CDS_PTR;


/*
 * DOS Directory entry for 'found' file - ALL DOS VERSIONS
 */
typedef struct
{
        char            file_name[11];
        uint8           file_attr;
        uint8           f1[10];
        uint32          file_time;
        uint16          start_sector;
        int32           file_size;
} DIRREC, far* DIRREC_PTR;

/*
 * Swappable DOS Area - DOS VERSION 3.xx
 */
typedef struct
{
        uint8           f0[12];
        uint8 far *     current_dta;
        uint8           f1[30];
        uint8           dd;
        uint8           mm;
        uint16          yy_1980;
        uint8           f2[96];
        char            file_name[128];
        char            file_name_2[128];
        SRCHREC         srchrec;
        DIRREC          dirrec;
        uint8           f3[81];
        char            fcb_name[11];
        uint8           f4;
        char            fcb_name_2[11];
        uint8           f5[11];
        uint8           srch_attr;
        uint8           open_mode;
        uint8           f6[48];
        uint8 far *     cdsptr;
        uint8           f7[72];
        SRCHREC         rename_srchrec;
        DIRREC          rename_dirrec;
} V3_SDA, far* V3_SDA_PTR;
```

```
/*
 * Swappable DOS Area - DOS VERSION 4.xx
 */
typedef struct
{
        uint8           f0[12];
        uint8 far *     current_dta;
        uint8           f1[32];
        uint8           dd;
        uint8           mm;
        uint16          yy_1980;
        uint8           f2[106];
        char            file_name[128];
        char            file_name_2[128];
        SRCHREC         srchrec;
        DIRREC          dirrec;
        uint8           f3[88];
        char            fcb_name[11];
        uint8           f4;
        char            fcb_name_2[11];
        uint8           f5[11];
        uint8           srch_attr;
        uint8           open_mode;
        uint8           f6[51];
        uint8 far *     cdsptr;
        uint8           f7[87];
        uint16          action_2E;
        uint16          attr_2E;
        uint16          mode_2E;
        uint8           f8[29];
        SRCHREC         rename_srchrec;
        DIRREC          rename_dirrec;
} V4_SDA, far* V4_SDA_PTR;


/*
 * DOS List of lists structure - DOS VERSIONS 3.1 thru 4
 * We don't need much of it.
 */
typedef struct
{
        uint8           f1[22];
        V3_CDS_PTR      cds_ptr;
        uint8           f2[7];
        uint8           last_drive;
} LOLREC, far* LOLREC_PTR;


/*
```

```
 * DOS 4.00 and above lock/unlock region structure
 * see lockfil() below (Thanks to Martin Westermeier.)
 */
typedef struct
{
        uint32          region_offset;
        uint32          region_length;
        uint8           f0[13];
        char            file_name[80]; // 80 is a guess
} LOCKREC, far* LOCKREC_PTR;


/*
 * The following structure is compiler specific, and maps
 * onto the registers pushed onto the stack for an interrupt
 * function.
 */
typedef struct
{
#ifdef __BORLANDC__
        uint16          bp, di, si, ds, es, dx, cx, bx, ax;
#else
        uint16          es, ds, di, si, bp, sp, bx, dx, cx, ax;
#endif
        uint16          ip,cs,flags;
} ALL_REGS;

#pragma pack()

/*
 * all the calls we need to support are in the range 0..2Eh
 * This serves as a list of the function types that we support
 */
#define         _inquiry                0x00
#define         _rd                     0x01
#define         _md                     0x03
#define         _shutdown               0x04
#define         _cd                     0x05
#define         _closefile              0x06
#define         _commitfile             0x07
#define         _readfile               0x08
#define         _writefile              0x09
#define         _lockfile               0x0A
#define         _unlockfile             0x0B
#define         _diskspace              0x0C
#define         _setfileattr            0x0E
#define         _getfileattr            0x0F
#define         _renamefile             0x11
#define         _deletefile             0x13
#define         _openfile               0x16
```

```
#define         _createfile             0x17
#define         _findfirst              0x1B
#define         _findnext               0x1C
#define         _seekfromend            0x21
#define         _unknown_fxn_2D      0x2D
#define         _specialopenfile        0x2E
#define         _unsupported         0xFF

#define ATTR_READONLY  0x01
#define ATTR_HIDDEN    0x02
#define ATTR_SYSTEM    0x04
#define ATTR_VOLUME    0x08
#define ATTR_SUBDIR    0x10
#define ATTR_ARCHIVE      0x20

#endif
```

Sourcefile: Redir.c

```
#pragma check_stack(off)
#include <stdlib.h>
#include <dos.h>
#include <memory.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <bios.h>
#include <time.h>

#include "systypes.h"
#include "vsprintf.h"
#include "tsr.h"
#include "ems.h"
#include "diskdev.h"
#include "debug.h"
#include "vfs.h"
#include "redir.h"

#define FILE_NOT_FOUND    2
#define PATH_NOT_FOUND    3
#define ACCESS_DENIED     5
#define NO_MORE_FILES    18

char *VolumeString="HD           ";
char *TsrName="MOUNTE2";
char *VersionString="1.0";

typedef struct
```

```
{
   uint8 far *   our_int2f_handler;
   uint8 far *   prev_int2f_handler;
   uint8            our_drive;
} user_data_t;

user_data_t      UserData;
int                 emskeep;


/* ***************************************************
   Constants and Macros
   *************************************************** */

#define     TRUE          1
#define     FALSE         0

#define     STACK_SIZE       1024
#define     FCARRY           0x0001
#define     MAX_FXN_NO       0x2E


char *usage_string = "Usage:   mounte2 [hda3] d:¥r¥n¥r¥n"
   "   hda3: linux partition on disk, first found if ommited ¥r¥n"
   "   d:  : drive on which the ext2fs should be mapped¥r¥n"
   "Options:¥r¥n"
   "   DC  : Debug to console (this is the default)¥r¥n"
   "   DEx : Debug to ems, use x pages (defaults to 1)¥r¥n"
   "   K   : Keep emspages after unloading¥r¥n"
   "   Lx  : Debug level x (defaults to 0)¥r¥n"
   "   M   : Maximum cache (512kb - 1mb)¥r¥n"
   "   Nx  : # of char's to make a file unique (defaults to 1)¥r¥n"
   "   SP  : Show partition tables¥r¥n"
   "   SF  : Show list of debug flags¥r¥n"
   "   U   : Unload latest version¥r¥n"
   "¥r¥n";



INTVECT  prev_int2f_handler;

/*
 * These all point to internael Dos datastructures
 */
char far *    cds_curpath;     /* ptr to current path in CDS                */
uint8 far *   sda_ptr;      /* ptr to SDA                                   */
char far *    sda_filename; /* ptr to 1st filename area in SDA              */
char far *    sda_fcbname;     /* ptr to 1st FCB-style name in SDA          */
uint8 far *sda_srch_attr;/* ptr to search attribute in SDA             */
SRCHREC_PTR sda_srchrec;     /* ptr to 1st Search Data Block in SDA     */
```

```
DIRREC_PTR    sda_dirrec;   /* ptr to 1st found dir entry area in SDA  */

/*
 * Other global data items
 */
ALL_REGS   GlobReg;      /* Global save area for all caller's regs  */
uint8         our_drive_no; /* A: is 1, B: is 2, etc.                    */
char          our_drive_str[3] = " :";  /* Our drive letter string        */
char far *    cds_path_root = "Linux :¥¥"; /* Root string for CDS         */
uint16        cds_root_size;/* Size of our CDS root string            */
LOLREC_PTR    lolptr;          /* pointer to List Of Lists                 */
uint16        dos_ss;          /* DOS's saved SS at entry                  */
uint16        dos_sp;          /* DOS's saved SP at entry                  */
uint16        our_sp;          /* SP to switch to on entry                 */
uint16        save_sp;      /* SP saved across internal DOS calls        */
int           filename_is_char_device;  /* generate_fcbname found character device name */
char          our_stack[STACK_SIZE]; /* our internal stack               */
uint16 far*stack_param_ptr; /* ptr to word at top of stack on entry */
int           curr_fxn;          /* Record of function in progress       */

uint8   fxnmap[] =
{
   _inquiry,           /* 0x00h */
   _rd,                /* 0x01h */
   _unsupported,    /* 0x02h */
   _md,                /* 0x03h */
   _shutdown,          /* 0x04h */
   _cd,                /* 0x05h */
   _closefile,      /* 0x06h */
   _commitfile,        /* 0x07h */
   _readfile,          /* 0x08h */
   _writefile,      /* 0x09h */
   _lockfile,          /* 0x0Ah */
   _unlockfile,        /* 0x0Bh */
   _diskspace,      /* 0x0Ch */
   _unsupported,    /* 0x0Dh */
   _setfileattr,    /* 0x0Eh */
   _getfileattr,    /* 0x0Fh */
   _unsupported,    /* 0x10h */
   _renamefile,        /* 0x11h */
   _unsupported,    /* 0x12h */
   _deletefile,        /* 0x13h */
   _unsupported,    /* 0x14h */
   _unsupported,    /* 0x15h */
   _openfile,          /* 0x16h */
   _createfile,        /* 0x17h */
   _unsupported,    /* 0x18h */
   _unsupported,    /* 0x19h */
   _unsupported,    /* 0x1Ah */
   _findfirst,      /* 0x1Bh */
```

```
   _findnext,           /* 0x1Ch */
   _unsupported,     /* 0x1Dh */
   _unsupported,     /* 0x1Eh */
   _unsupported,     /* 0x1Fh */
   _unsupported,     /* 0x20h */
   _seekfromend,     /* 0x21h */
   _unsupported,     /* 0x22h */
   _unsupported,     /* 0x23h */
   _unsupported,     /* 0x24h */
   _unsupported,     /* 0x25h */
   _unsupported,     /* 0x26h */
   _unsupported,     /* 0x27h */
   _unsupported,     /* 0x28h */
   _unsupported,     /* 0x29h */
   _unsupported,     /* 0x2Ah */
   _unsupported,     /* 0x2Bh */
   _unsupported,     /* 0x2Ch */
   _unknown_fxn_2D,  /* 0x2Dh */
   _specialopenfile /* 0x2Eh */
};



/*************************************************
 * Internal Dos Calls
 *************************************************/
void set_sft_owner(SFTREC_PTR sft)
   {
   _asm {
      push es
        push di
        les di, sft
        mov save_sp, sp     /* Save current stack pointer. */
        cli
        mov ss, dos_ss      /* Establish DOS's stack, current */
        mov sp, dos_sp      /* when we got called. */
        sti
        mov ax, 0x120c      /* Claim file as ours. */
        push ds             /* It needs DS to be DOS's DS, for DOS 5.0 */
        mov ds, GlobReg.ds
        int 0x2F
        pop bx              /* Restore DS */
        mov ds, bx          /* Restore SS (same as DS) */
        cli
        mov ss, bx
        mov sp, save_sp     /* and stack pointer (which we saved). */
        sti
        pop di
        pop es
    }
```

```
}

/*
 * Does sda_fcbname point to a device name?
 */
int is_a_character_device(uint16 dos_ds)
{
   _asm {
      mov ax, 0x1223   /* Search for device name. */
      push ds          /* It needs DS to be DOS's DS, for DOS 5.0 */
      mov ds, dos_ds
      int 0x2F
      pop ds           /* Restore DS */
      jnc is_indeed
   }
   return FALSE;
is_indeed:
   return TRUE;

}



/*
 * Some frequently used service routines
 */

/*
 * Fail, print message, exit to DOS
 */
void failprog(char * msg)
{
   tsr_print_string((uint8 far *) msg, TRUE);
   exit(1);
}


void get_fcbname_from_path(char far* path, char far* fcbname)
{
   int i;

   _fmemset(fcbname, ' ', 11);
   for (i = 0; *path; path++)
      if (*path == '.')
         i = 8;
      else
         fcbname[i++] = *path;
}
```

```
/*
 * Fail the current redirector call with the supplied error number, i.e.
 * set the carry flag in the returned flags, and set ax=error code
 */

void fail(uint16 err)
{
   GlobReg.flags=(GlobReg.flags | FCARRY);
   GlobReg.ax = err;
}


void succeed(void)
{
   GlobReg.flags=(GlobReg.flags & ~FCARRY);
   GlobReg.ax = 0;
}


/*
 * Deal with differences in DOS version once, and set up a set
 * of absolute pointers
 */

void set_up_pointers(void)
{
   if (_osmajor==3)
   {
      sda_fcbname=((V3_SDA_PTR) sda_ptr)->fcb_name;
      sda_filename=((V3_SDA_PTR) sda_ptr)->file_name + cds_root_size - 1;

      sda_srchrec= &((V3_SDA_PTR) sda_ptr)->srchrec;
      sda_dirrec=&((V3_SDA_PTR) sda_ptr)->dirrec;
      sda_srch_attr=&((V3_SDA_PTR) sda_ptr)->srch_attr;
   }
   else
   {
      sda_fcbname=((V4_SDA_PTR) sda_ptr)->fcb_name;
      sda_filename=((V4_SDA_PTR) sda_ptr)->file_name + cds_root_size - 1;
      sda_srchrec=&((V4_SDA_PTR) sda_ptr)->srchrec;
      sda_dirrec=&((V4_SDA_PTR) sda_ptr)->dirrec;
      sda_srch_attr=&((V4_SDA_PTR) sda_ptr)->srch_attr;
   }
}



/*
 * This function should not be necessary. DOS usually generates an FCB
 * style name in the appropriate SDA area. However, in the case of
```

```
 * user input such as 'CD ..' or 'DIR ..' it leaves the fcb area all
 * spaces. So this function needs to be called every time. Its other
 * feature is that it uses an internal DOS call to determine whether
 * the filename is a DOS character device. We will 'Access deny' any
 * use of a char device explicitly directed to our drive
 */

void generate_fcbname(uint16 dos_ds)
{
   get_fcbname_from_path((char far*) (_fstrrchr(sda_filename, '¥¥') + 1),
      sda_fcbname);

   filename_is_char_device = is_a_character_device(dos_ds);
}


/*
 * Does the supplied string contain a wildcard '?'
 */
int contains_wildcards(char far* path)
{
   int i;

   for (i = 0; i < 11; i++)
      if (path[i] == '?')
         return TRUE;
   return FALSE;
}

/*
 * Get DOS version, address of Swappable DOS Area, and address of
 * DOS List of lists. We only run on versions of DOS >= 3.10, so
 * fail otherwise
 */

void get_dos_vars(void)
{
   uint16 segmnt;
   uint16 ofset;

   if ((_osmajor < 3) || ((_osmajor == 3) && (_osminor < 10)))
      failprog("Unsupported DOS Version");

   _asm {
      push ds
      push es
      mov ax, 5d06h;/* Get SDA pointer */
      int 21h;
      mov segmnt, ds
      mov ofset, si
```

```
      pop es
      pop ds
   }
   sda_ptr = MK_FP(segmnt, ofset);

   _asm {
      push ds
      push es
      mov ax, 5200h;/* Get Lol pointer */
      int 21h;
      mov segmnt, es
      mov ofset, bx
      pop es
      pop ds
      }
   lolptr = (LOLREC_PTR) MK_FP(segmnt, ofset);
}


int is_call_for_us(uint16 es, uint16 di, uint16 ds)
{
   uint8 far * p;
   int ret = 0xFF;

   filename_is_char_device = 0;

   /*
    * Note that the first 'if' checks for the bottom 6 bits
    * of the device information word in the SFT. Values > last drive
    * relate to files not associated with drives, such as LAN Manager
    * named pipes (Thanks to Dave Markun).
    */

   if ((curr_fxn >= _closefile && curr_fxn <= _unlockfile)
      || (curr_fxn == _seekfromend)
      ||  (curr_fxn == _unknown_fxn_2D) )
   {
      ret = ((((SFTREC_PTR) MK_FP(es,di))->dev_info_word & 0x3F)
         == our_drive_no);
   }
   else
   {
      if (curr_fxn ==_inquiry || curr_fxn==_shutdown)  // 2F/1100 -- succeed automatically
           ret = TRUE;
      else
      {
         if (curr_fxn == _findnext)      // Find Next
         {
            SRCHREC_PTR psrchrec;  // check search record in SDA
            if (_osmajor==3)
```

```
            psrchrec=&(((V3_SDA_PTR) sda_ptr)->srchrec);
         else
            psrchrec=&(((V4_SDA_PTR) sda_ptr)->srchrec);
         return (int) ((psrchrec->drive_no & (uint8) 0x40) &&
            ((psrchrec->drive_no & 0x1F) == our_drive_no));
      }
      if (_osmajor==3)
         p=((V3_SDA_PTR) sda_ptr)->cdsptr; // check CDS
      else
         p=((V4_SDA_PTR) sda_ptr)->cdsptr;

      if (_fmemcmp(cds_path_root, p, cds_root_size) == 0)
      {
         // If a path is present, does it refer to a character device
         if (curr_fxn != _diskspace)
            generate_fcbname(ds);
         return TRUE;
      }
      else
         return FALSE;
   }
  }
  return ret;
}


/*
 * Check to see that we are allowed to install
 */

void is_ok_to_load(void)
{
  _asm {
    mov ax, 1100h;
    int 2fh;
    cmp ax,1
    jne go_forward;
  }
  failprog("Not OK to install a redirector...");

go_forward:
  return;
}


/*
 * This is where we do the initializations of the DOS structures
 * that we need in order to fit the mould
 */
```

```
void set_up_cds(void)
{
   V3_CDS_PTR our_cds_ptr;

   our_cds_ptr = lolptr->cds_ptr;
   if (_osmajor == 3)
      our_cds_ptr = our_cds_ptr + our_drive_no;
   else
   {
      V4_CDS_PTR t = (V4_CDS_PTR) our_cds_ptr;
      t = t + our_drive_no;
      our_cds_ptr = (V3_CDS_PTR) t;
   }

   if (our_drive_no >= lolptr->last_drive)
      failprog("Drive letter higher than last drive.");

   /*
    * Check that this drive letter is currently invalid (not in use
    * already)
    * 0xc000 tests both physical and network bits at same time
    */

   if ((our_cds_ptr->flags & 0xc000) != 0)
      failprog("Drive already assigned...");

   /*
    * Set Network+Physical+NotRealNetworkDrive bits on, and
    * establish our 'root'
    */
   our_cds_ptr->flags |= 0xc080;
   cds_root_size = _fstrlen(cds_path_root);
   _fstrcpy(our_cds_ptr->current_path, cds_path_root);
   our_cds_ptr->current_path[_fstrlen(our_cds_ptr->current_path) - 3] =
      (char) ('A'+ our_drive_no);
   _fstrcpy(cds_path_root, our_cds_ptr->current_path);
   cds_curpath = our_cds_ptr->current_path;
   our_cds_ptr->root_ofs = _fstrlen(our_cds_ptr->current_path) - 1;
   cds_curpath += our_cds_ptr->root_ofs;
}



/*
 * All the redirector routines, remember, for the moment we're
 * a read-only device
 */


/*
```

```
 * Respond that it is OK to load another redirector
 */

void redir_inquiry(void)           // subfunc 01
{
   GlobReg.ax = 0x00FF;
}

void redir_readonlydevice(void)    // subfunc 01
{
   fail(ACCESS_DENIED);
}

void redir_findnext(void)          // subfunc 1C
{

   dprintf(D_REDIR,"finding next...¥n");
   if (!(vfs_findnext(0, sda_srchrec, sda_dirrec)))
   {
      fail(NO_MORE_FILES);
      return;
   }
   succeed();
}

/*
 * FindFirst - subfunction 1Bh
 */

void redir_findfirst(void)
{

   dprintf(D_REDIR,"findfirst:%fs, mask:%x¥n",sda_filename, (unsigned) *sda_srch_attr);
   /*
    * Handle volumesearch, must be in root
    */

   if (*sda_srch_attr == ATTR_VOLUME)
   {
      if(sda_filename != _fstrrchr(sda_filename,'¥¥'))
      {
         fail(PATH_NOT_FOUND);
         return;
      }
      else
      {
         fail(PATH_NOT_FOUND);
         return;
      }
   }
```

```
   if (!vfs_initsearch(0, sda_filename))
   {
      fail(PATH_NOT_FOUND);
      return;
   }

   _fmemcpy(&sda_srchrec->srch_mask, sda_fcbname, 11);

   sda_srchrec->dir_entry_no = 0;
   sda_srchrec->attr_mask = *sda_srch_attr;
   sda_srchrec->drive_no = (uint8) (our_drive_no | 0xC0);


   redir_findnext();

   /*
    * No need to check GlobReg.flags & FCARRY; if ax is 18,
    * FCARRY must have been set.
    */

   if (GlobReg.ax == 18)
      GlobReg.ax = 2;     // make findnext error code suitable to findfirst
}




/*
 * Change Directory - subfunction 05h
 */

void redir_cd(void)
{
   dprintf(D_REDIR,"cd to:%fs¥n",sda_filename);
   if ((*sda_filename != '¥¥') || (*(sda_filename + 1)))
   {
      if (contains_wildcards(sda_fcbname))
      {
         fail(PATH_NOT_FOUND);
         return;
      }

      *sda_srch_attr = 0x10;
      redir_findfirst();
      if (GlobReg.ax || (! (sda_dirrec->file_attr & 0x10)))
      {
         fail(PATH_NOT_FOUND);
         return;
      }
```

```
   }
   _fstrcpy(cds_curpath, sda_filename);
}


/*
 * Close File - subfunction 06h
 */

void redir_closefile(void)
{
   SFTREC_PTR p = (SFTREC_PTR) MK_FP(GlobReg.es, GlobReg.di);

   dprintf(D_REDIR,"close:%fs¥n",p->file_name);
   if (p->handle_count)      /* If handle count not 0, decrement it */
      --p->handle_count;
}


/*
 * Read from File - subfunction 08h
 * For version that handles critical errors,
 * see Undocumented DOS, 2nd edition, chapter 8
 */

void redir_readfile(void)
{
   SFTREC_PTR p = (SFTREC_PTR) MK_FP(GlobReg.es, GlobReg.di);

   dprintf(D_REDIR,"readfile:%fs, file_size:%lu, file_pos:%lu count:%u¥n",p->file_name,p-
>file_size,p->file_pos,GlobReg.cx);
   if (p->open_mode & 1)
   {
      fail(ACCESS_DENIED);
      return;
   }

   if ((p->file_pos + GlobReg.cx) > p->file_size)
      GlobReg.cx = (uint16) (p->file_size - p->file_pos);

   if (! GlobReg.cx)
      return;

         // V3_SDA_PTR cast is save: current_dta has fixed offset in SDA
   GlobReg.cx=vfs_readfile(0,p, GlobReg.cx, ((V3_SDA_PTR) sda_ptr)->current_dta);
}


/*
 * Get Disk Space - subfunction 0Ch
```

```
 */

void redir_diskspace(void)
{
   dprintf(D_REDIR,"diskspace¥n");
   GlobReg.ax=vfs_secpercluster(0);
   GlobReg.bx=vfs_clusters(0);
   GlobReg.cx=vfs_sectorsize(0);;
   GlobReg.dx=vfs_freeclusters(0);;
}


/*
 * Get File Attributes - subfunction 0Fh
 */

void redir_getfileattr(void)
{
   dprintf(D_REDIR,"getfileattrib:%fs¥n",sda_filename);
   if (contains_wildcards(sda_fcbname))
   {
      fail(FILE_NOT_FOUND);
      return;
   }

   *sda_srch_attr = 0x3f;
   redir_findfirst();
   if (GlobReg.ax)
      return;

   GlobReg.ax = (uint16) sda_dirrec->file_attr;
         /* should we set BX:DI --> filesize ?? */
}



/*
 * Open Existing File - subfunction 16h
 */

void redir_openfile(void)
{
   SFTREC_PTR p=(SFTREC_PTR) MK_FP(GlobReg.es, GlobReg.di);

   dprintf(D_REDIR,"open:%fs¥n",sda_fcbname);

   if (contains_wildcards(sda_fcbname))
   {
      fail(PATH_NOT_FOUND);
      return;
```

```
   }

   *sda_srch_attr = 0x27;
   redir_findfirst();

/*
 * If we have a FILE_NOT_FOUND, we do not return immediately, but
 * give the VFS a chance to handle 'special files'
 */

   if (GlobReg.ax)
      return;

   vfs_fill_sft(0,p, sda_dirrec, sda_fcbname);

    if (p->open_mode & 0x8000)
         /* File is being opened via FCB */
         p->open_mode |= 0x00F0;
       else
         p->open_mode &= 0x000F;

   /*
    * Mark file as being on network drive, unwritten to
    */

    p->dev_info_word = (uint16) (0x8040 | (uint16) our_drive_no);
    p->dev_drvr_ptr = NULL;
   p->file_pos=0l;

    dprintf(D_REDIR,"fill_sft opening:%11fs ,mode:%x¥n",
         p->file_name,p->open_mode);

   if(p->open_mode & 0x8000)
      set_sft_owner(p);
}

void redir_createfil(void)
{
   dprintf(D_REDIR,"create file:%fs¥n",sda_fcbname);
   fail(ACCESS_DENIED);
   return;
}


/* Special Multi-Purpose Open File - subfunction 2Eh */

#define CREATE_IF_NOT_EXIST  0x10
#define OPEN_IF_EXISTS       0x01
#define REPLACE_IF_EXISTS    0x02
```

```
void redir_special_openfile(void)
{
   SFTREC_PTR p = (SFTREC_PTR) MK_FP(GlobReg.es,GlobReg.di);
   unsigned open_mode, action;

   open_mode = ((V4_SDA_PTR) sda_ptr)->mode_2E & 0x7f;
   action = ((V4_SDA_PTR) sda_ptr)->action_2E;
   p->open_mode = open_mode;
   dprintf(D_REDIR,"special open:%fs, mode:%u, action:%u¥n",sda_fcbname,open_mode,action);

   if (contains_wildcards(sda_fcbname))
   {
      fail(PATH_NOT_FOUND);
      return;
   }

   *sda_srch_attr = 0x3f;
   redir_findfirst();
   if ((GlobReg.flags & FCARRY) && (GlobReg.ax != 2))
      return;


   if (GlobReg.ax)
   {
      /*
       * File does not exist
       */

      if (action & 0xF0)
      {
         fail(ACCESS_DENIED);         // attempt to create
         return;
      }
      else
         return;       // we had a fail(FILE_NOT_FIND)
   }
   else
   {
      /*
       * File exists
       */

      if ((action &0x0F) != 1)
      {
         fail(ACCESS_DENIED);      // attempt to replace file
         return;
      }
   }

   vfs_fill_sft(0,p, sda_dirrec, sda_fcbname);
```

```
   if (p->open_mode & 0x8000)
         /* File is being opened via FCB */
      p->open_mode |= 0x00F0;
   else
      p->open_mode &= 0x000F;

   /*
    * Mark file as being on network drive, unwritten to
    */

   p->dev_info_word = (uint16) (0x8040 | (uint16) our_drive_no);
   p->file_pos = 0l;
   p->dev_drvr_ptr = NULL;

   dprintf(D_REDIR,"fill_sft special opening:%11fs ,mode:%x¥n",
         p->file_name,p->open_mode);

   if(p->open_mode & 0x8000)
      set_sft_owner(p);

   succeed();
}


/*
 * This function is never called! DOS fiddles with position internally
 */

void redir_seekfromend(void)
{
   int32 seek_amnt;
   SFTREC_PTR p;

   /* But, just in case... */
   seek_amnt = -1L * (((int32) GlobReg.cx << 16) + GlobReg.dx);
   p = (SFTREC_PTR) MK_FP(GlobReg.es,GlobReg.di);
   if (seek_amnt > p->file_size)
      seek_amnt = p->file_size;

   p->file_pos = p->file_size - seek_amnt;
   GlobReg.dx = (uint16) (p->file_pos >> 16);
   GlobReg.ax = (uint16) (p->file_pos & 0xFFFF);
}

void redir_closeall()
{
}

void redir_unknown_fxn_2D()
```

```
{
   dprintf(D_REDIR,"Unknown function_2D:%u¥n",curr_fxn);
   GlobReg.ax = 2;
   /* Only called in v4.01, this is what MSCDEX returns */
}

void redir_shutdown(void)
{

   dprintf(D_SYSTEM,"Preparing shutdown....¥n");
   vfs_shutdown(0);
   // debug_shutdown(); we're cleaned up by the unload process
}

/*
 * A placeholder
 */

void redir_unsupported(void)
{
   dprintf(D_REDIR,"Unsupported function:%u¥n",curr_fxn);
   return;
}

PROC dispatch_table[]=
{
   redir_inquiry,            /* 0x00h */
   redir_readonlydevice,     /* 0x01h */
   redir_unsupported,        /* 0x02h */
   redir_readonlydevice,     /* 0x03h */
   redir_shutdown,              /* 0x04h */
   redir_cd,                    /* 0x05h */
   redir_closefile,          /* 0x06h */
   redir_readonlydevice,     /* 0x07h */
   redir_readfile,              /* 0x08h */
   redir_readonlydevice,     /* 0x09h */
   redir_unsupported,        /* 0x0Ah */
   redir_unsupported,        /* 0x0Bh */
   redir_diskspace,          /* 0x0Ch */
   redir_unsupported,        /* 0x0Dh */
   redir_readonlydevice,     /* 0x0Eh */
   redir_getfileattr,        /* 0x0Fh */
   redir_unsupported,        /* 0x10h */
   redir_readonlydevice,     /* 0x11h */
   redir_unsupported,        /* 0x12h */
   redir_readonlydevice,     /* 0x13h */
   redir_unsupported,        /* 0x14h */
   redir_unsupported,        /* 0x15h */
   redir_openfile,              /* 0x16h */
   redir_readonlydevice,     /* 0x17h */
```

```
   redir_unsupported,        /* 0x18h */
   redir_unsupported,        /* 0x19h */
   redir_unsupported,        /* 0x1Ah */
   redir_findfirst,          /* 0x1Bh */
   redir_findnext,              /* 0x1Ch */
   redir_unsupported,        /* 0x1Dh */
   redir_unsupported,        /* 0x1Eh */
   redir_unsupported,        /* 0x1Fh */
   redir_unsupported,        /* 0x20h */
   redir_seekfromend,        /* 0x21h */
   redir_unsupported,        /* 0x22h */
   redir_unsupported,        /* 0x23h */
   redir_unsupported,        /* 0x24h */
   redir_unsupported,        /* 0x25h */
   redir_unsupported,        /* 0x26h */
   redir_unsupported,        /* 0x27h */
   redir_unsupported,        /* 0x28h */
   redir_unsupported,        /* 0x29h */
   redir_unsupported,        /* 0x2Ah */
   redir_unsupported,        /* 0x2Bh */
   redir_unsupported,        /* 0x2Ch */
   redir_unsupported,        /* 0x2Dh */
   redir_special_openfile    /* 0x0Eh */
   };




/*
 * This is the main entry point for the redirector. It assesses if
 * the call is for our drive, and if so, calls the appropriate routine.
 * On return, it restores the (possibly modified) register values.
 */

void interrupt far redirector(ALL_REGS entry_regs)
{
   static uint16 save_bp;
   static uint16 StkPtr;

   _asm STI;

   if (((entry_regs.ax >> 8) != (uint8) 0x11) ||
      ((uint8) entry_regs.ax > MAX_FXN_NO))
         goto chain_on;

   curr_fxn = fxnmap[(uint8) entry_regs.ax];

   if ((curr_fxn == _unsupported) ||
      (! is_call_for_us(entry_regs.es, entry_regs.di, entry_regs.ds)))
      goto chain_on;
```

```
  /* Set up our copy of the registers */
  GlobReg.ax=entry_regs.ax;
  GlobReg.bx=entry_regs.bx;
  GlobReg.cx=entry_regs.cx;
  GlobReg.dx=entry_regs.dx;
  GlobReg.es=entry_regs.es;
  GlobReg.ds=entry_regs.ds;
  GlobReg.bp=entry_regs.bp;
  GlobReg.sp=entry_regs.sp;
  GlobReg.si=entry_regs.si;
  GlobReg.di=entry_regs.di;
  GlobReg.ip=entry_regs.ip;
  GlobReg.cs=entry_regs.cs;
  GlobReg.flags=entry_regs.flags;

/*
 * AAAARRRGGGGHHH, somehow this struct copy causes a hang, donno why
 */
  //GlobReg=entry_regs;

  /*
   * Save ss:sp and switch to our internal stack. We also save bp
   * so that we can get at any parameter at the top of the stack
   * (such as the file attribute passed to subfxn 17h).
   */

  _asm cli
  _asm mov dos_ss, ss;
  _asm mov save_bp, bp;
  _asm sti

/*
 * Make sure the SP is an even address
 */

  StkPtr=(unsigned) our_stack + STACK_SIZE -2;
  if (StkPtr & 1u)
     StkPtr--;
  _asm {
     cli
     mov dos_sp, sp;
     mov ax, ds
     mov ss, ax       // New stack segment is in Data segment.
     //mov sp, offset our_stack + STACK_SIZE -2
     mov sp, StkPtr
     sti
  }

  // Expect success!
```

```
   succeed();

   /*
    * Call the appropriate handling function unless we already know we
    * need to fail
    */
   if (filename_is_char_device)
      fail(ACCESS_DENIED);
   else
   {
      vfs_savecontext();
      dispatch_table[curr_fxn]();
      vfs_restorecontext();
   }

   // Switch the stack back
   _asm {
      cli;
      mov ss, dos_ss;
      mov sp, dos_sp;
      sti;
   }

   /*
    * put the possibly changed registers back on the stack, and return
    * entry_regs=GlobReg;
    */
   entry_regs.ax=GlobReg.ax;
   entry_regs.bx=GlobReg.bx;
   entry_regs.cx=GlobReg.cx;
   entry_regs.dx=GlobReg.dx;
   entry_regs.es=GlobReg.es;
   entry_regs.ds=GlobReg.ds;
   entry_regs.bp=GlobReg.bp;
   entry_regs.sp=GlobReg.sp;
   entry_regs.si=GlobReg.si;
   entry_regs.di=GlobReg.di;
   entry_regs.ip=GlobReg.ip;
   entry_regs.cs=GlobReg.cs;
   entry_regs.flags=GlobReg.flags;
   return;

   // If the call wasn't for us, we chain on.
chain_on:
   _chain_intr(prev_int2f_handler);
}


/*
 * Fill in our UserData, upon unloading were are called back with this
```

```
 * information
 */

void prepare_for_tsr()
{
   prev_int2f_handler=_dos_getvect(0x2f);
   UserData.prev_int2f_handler=(uint8 far *) prev_int2f_handler;
   UserData.our_int2f_handler=(uint8 far *) redirector;
   UserData.our_drive=our_drive_no;
   _dos_setvect(0x2f,redirector);
}



/*
 * This is called by the tsr_unload routine with info pointing to the
 * the the signature_t of the running trs. If is has been superceded
 * return 0
 */
int unload_redir(signature_t far * sigrec)
{
   INTVECT    p_vect;
   user_data_t   far *user_data=(user_data_t far *) sigrec->user_data;
   V3_CDS_PTR    cds_ptr;


/*
 * give the running TSR a change to shutdown properly
 */
   _asm {
      mov   ah,0x11;
      mov al,_shutdown;
      int 2Fh;
   }

   p_vect=_dos_getvect(0x2f);

   /*
    * Check that a subsequent TSR hasn't taken over Int 2Fh
    */

   if (user_data->our_int2f_handler != (uint8 far *) p_vect)
   {
      tsr_print_string("Interrupt 2F has been superceded...",1);
      return 0;
   }

   /*
    * Fix int2f chain
    */
```

```
   p_vect=(INTVECT) user_data->prev_int2f_handler;
   _dos_setvect(0x2f,p_vect);

   our_drive_no=user_data->our_drive;
   cds_ptr=lolptr->cds_ptr;
   if (_osmajor == 3)
      cds_ptr+=our_drive_no;
   else
   {
      V4_CDS_PTR t=(V4_CDS_PTR) cds_ptr;
      t+=our_drive_no;
      cds_ptr=(V3_CDS_PTR) t;
   }

   /*
    * switch off the Network and Physical bits for the drive,
    * rendering it invalid.
    */

   cds_ptr->flags=cds_ptr->flags & 0x3fff;
   our_drive_str[0]=(char) (our_drive_no + 'A');
   tsr_print_string(our_drive_str,FALSE);
   tsr_print_string(" is now invalid.",TRUE);
   if (!emskeep)
      debug_shutdown();
   return 1;
}


int _cdecl main(uint16 argc , char **argv)
{
   int disk, partition, dout, dlevel, dpages, showpart, showdlist;
   int   maxcache, unload, unique;

   dlevel=showpart=showdlist=0,maxcache=emskeep=unload=0;
   unique=dpages=1;
   dout=DOUT_CONSOLE;
   disk=partition=-1;

   for (argv++; *argv; argv++)
   {
      switch (**argv)
      {
         case '-':
         case '/':
            (*argv)++;
            switch (toupper(**argv))
            {
               case 'U':
```

```
            unload=1;
            break;
         case 'N':
            unique=simple_atoi((*argv)+1);
            break;
         case 'D':
            (*argv)++;
            switch(toupper(**argv))
            {
               case 'C':
                  dout=DOUT_CONSOLE;
                  break;
               case 'E':
                  dout=DOUT_EMS;
                  dpages=simple_atoi((*argv)+1);
                  if (!dpages)
                     dpages++;     // at least 1 page
                  break;
               default:
                  dout=DOUT_CONSOLE;
                  dlevel=0;
                  break;
            }
            break;
         case 'L':
            dlevel=simple_atoi((*argv)+1);
            break;
         case 'M':
            maxcache=1;
            break;
         case 'K':
            emskeep=1;
            break;
         case 'S':
            (*argv)++;
            switch(toupper(**argv))
            {
               case 'P':
                  showpart=1;
                  break;
               case 'F':
                  showdlist=1;
                  break;
               default:
                  tsr_print_string("Unrecognized parameter.", TRUE);
                  return 0;
            }
            break;
         default:
            tsr_print_string("Unrecognized parameter.", TRUE);
```

```
                return 0;
            }
            break;
         case 'h':
         case 'H':
            if (*((*argv)+1)==':')
            {
               our_drive_str[0]='H';
               break;
            }

            disk=toupper(*((*argv)+2))-'A';
            partition=*((*argv)+3)-'0';
            if (disk<0 || disk >MAX_HD ||
               partition<0 || partition>MAX_LDEV)
            {
               tsr_print_string("What partition?¥r¥nUse option SP for partition
tables¥r¥n", TRUE);
               return 0;
            }
            break;
         default:
            our_drive_str[0] = **argv;
      }
   }

   if (unload)
   {
      get_dos_vars();
      tsr_unload_latest(TsrName,unload_redir);
      return 0;
   }

   if (!emsPresent())
   {
      tsr_print_string("Ems not present",TRUE);
      return 0;
   }
   tsr_print_string("MOUNTE2 v.",0);
   tsr_print_string(VersionString,1);

   if (showpart)
   {
      debug_init(D_PARTITION,DOUT_CONSOLE,0);
      pdev_readpartitions();
      return 0;
   }
   if (showdlist)
   {
      debug_init(0,DOUT_CONSOLE,0);
```

```
      debug_showlist();
      dprintf(D_ALWAYS,"¥nUse binary | to combine debuglevels¥n");
      return 0;
   }

      // check that it's a valid drive letter
   if (our_drive_str[0] == ' ')
   {
      tsr_print_string(usage_string, TRUE);
      return 0;
   }

   our_drive_str[0] &= ~0x20;
   our_drive_no = (uint8) (our_drive_str[0] - 'A');
   if (our_drive_no > 25)
   {
      tsr_print_string(usage_string, TRUE);
      return 0;
   }

/*
 * Let's first check that we are 'allowed' to load,
 * and the desired drive is not in use
 */
   is_ok_to_load();
   get_dos_vars();
   set_up_cds();
   set_up_pointers();

/*
 * Prepare debugging and read partitions tables
 */

   debug_init(dlevel, dout, dpages);
   dprintf(D_EMS,"Ems v%s: tot pages:%u, free: %u¥n", emsGetVersion(),
emsNumTotPages(),emsNumFreePages());
   pdev_readpartitions();

   if (!(vfs_setuppartition(&disk,&partition,maxcache,unique)))
      return 0;

   dout=debug_setout(DOUT_CONSOLE);
   dprintf(D_ALWAYS,"hd%c%c installed as %s¥n",disk+'a',partition+'0', our_drive_str);


/*
 * Let's hit it Stan!
 */

   prepare_for_tsr();                // sets UserData and chains int2f
```

```
   debug_setout(dout);
   tsr_go_resident(TsrName,&UserData);

   return 0;
}
```

Sourcefile: Systypes.h

```
#ifndef __SYSTYPES_H
#define __SYSTYPES_H

typedef unsigned char  uint8;
typedef unsigned int   uint16;
typedef unsigned long  uint32;
typedef signed   char  int8;
typedef signed   int   int16;
typedef signed   long  int32;


#endif     // __SYSTYPES_H
```

Sourcefile: Tsr.h

```
#ifndef __TSR_H
#define __TSR_H

#include "systypes.h"

#define RUNNING_AS_TSR
#ifndef MK_FP
#define MK_FP(a,b)  ((void far *)(((uint32)(a) << 16) | (b)))
#endif

#define MAX_SIG 10
#define MAX_USR 12

#pragma pack(1)

/*
 * TSR signature and unload info structure
 */

typedef struct
{
     uint8         cmdline_len;
     char          signature[MAX_SIG]; /* The TSR's signature string */
```

```
     uint16        psp;                /* This instance's PSP */
     char          user_data[MAX_USR];  /* Do with is as you please */
} signature_t;
#pragma pack()

typedef void (interrupt far* INTVECT)();
typedef int (unload_callbackfunc)(signature_t far *);


void*    tsr_malloc(unsigned size);
void*    tsr_calloc(unsigned items, unsigned item_size);
void     tsr_print_string(char far *string, int newline);
void     tsr_go_resident(char *signame, void *user_data);
void     tsr_unload_latest(char *name, unload_callbackfunc f);

#endif     // __TSR_H
```

Sourcefile: Tsr.c

```
#pragma check_stack(off)
#include <string.h>
#include <stdlib.h>
#include <dos.h>
#ifdef __BORLANDC__
#include <alloc.h>
#include <mem.h>
#else
#include <malloc.h>
#include <memory.h>
#endif

#include "tsr.h"
signature_t   sigrec = { 8, "", 0, "" };   /* our signature record */

unsigned         MemTop=0;
unsigned         Alloc_Size=0;



void *tsr_malloc(unsigned sz)
{
   return calloc(1,sz);
}

void * tsr_calloc(unsigned items, unsigned item_size)
{
   void       *ptr;
```

```
   unsigned sz;

   ptr=calloc(items,item_size);
   sz=items*item_size;
   Alloc_Size+=sz;
   MemTop=(((unsigned) ptr)+sz > MemTop)? ((unsigned) ptr)+sz : MemTop;

   return ptr;
}


void tsr_go_resident(char *signame, void *user_data)
{
   uint16 tsr_paras;   // Paragraphs to terminate and leave resident.
   uint16 highest_seg,dummy;

   uint8 far * buf;
   int i;


   if (!signame)
      return ;

   /*
    * Find ourselves a free interrupt to call our own. Without it,
    * we can still load, but a future invocation of Phantom with -U
    * will not be able to unload us.
    */

   for (i = 0x60; i < 0x67; i++)
   {
      int32 far* p;
      p = (int32 far *) MK_FP(0, ((uint16) i * 4));
      if (*p == 0L)
         break;
   }


   if (i == 0x67)
   {
      tsr_print_string("No user intrs available. TSR is not unloadable..",
         1);
      return;
   }

   /*
    * Our new found 'user' interrupt will point at the command line
    * area of our PSP. Complete our signature record, put it into the
    * command line, then go to sleep.
    */
```

```
   _dos_setvect(i, (INTVECT) (buf = MK_FP(_psp, 0x80)));

   strncpy(sigrec.signature,signame,MAX_SIG);
   memcpy(sigrec.user_data,user_data,MAX_USR);
   sigrec.psp = _psp;
   *((signature_t far *) buf) = sigrec;

   _asm mov highest_seg, ds;

   tsr_paras= highest_seg + 1 + ( (MemTop+16) >>4) - _psp;
   _dos_setblock(tsr_paras,_psp,&dummy);
   _dos_keep(0, tsr_paras);

   // we're gone....
}

void tsr_unload_latest(char *name, unload_callbackfunc unload_func)
{
   int                i;
   INTVECT          p_vect;
   signature_t far *sig_ptr;
   uint16             psp;
   static uint16    save_ss, save_sp;

   /*
    * Note that we step backwards to allow unloading of Multiple copies
    * in reverse order to loading, so that the Int 2Fh chain remains
    * intact.
    */
   for (i = 0x66; i >= 0x60; i--)
   {
      int32 far* p;
      p = (int32 far *) MK_FP(0, ((uint16) i * 4));
      sig_ptr = (signature_t far *) *p;
      if (_fstrncmp(sig_ptr->signature, name,MAX_SIG) == 0)
         break;
   }

   if (i == 0x5f)
   {
      tsr_print_string(name,0);
      tsr_print_string(" was not loaded.",1);
      return;
   }

   if (unload_func && !unload_func(sig_ptr))
   {
      tsr_print_string("TSR is not unloaded.",1);
      return;
```

```
   }

   p_vect = _dos_getvect(i);
   psp = ((signature_t far *) p_vect)->psp;

   /*
    * Use the recommended switch PSP and Int 4Ch method of
    * unloading the TSR (see TSRs chapter of Undocumented DOS).
    */

   _asm {
      // Save some registers
      push  es;
      push  ds;
      push  si;
      push  di;
      push  bp;
      // Set resident program's parent PSP to us.
      mov      es, psp;
      mov      bx, 0x16;
      mov      ax, _psp;
      mov      es:[di], ax;
      mov      di, 0x0a;
      // Set resident program PSP return address to exit_ret;
      mov      ax, offset exit_ret;
      stosw;
      mov      ax, cs;
      stosw;
      mov      bx, es;
      // Set current PSP to resident program
      mov      ah, 0x50;
      int      21h;
      // Save SS:SP
      mov      ax, seg save_ss;
      mov      ds, ax;
      mov      save_ss, ss;
      mov      save_sp, sp;
      // and terminate
      mov      ax, 0x4c00;
      int      21h;
   }
exit_ret:
   _asm {
      // We should arrive back here - restore SS:SP
      mov      ax, seg save_ss;
      mov      ds, ax;
      mov      ss, save_ss;
      mov      sp, save_sp;
      // restore the registers
      pop      bp;
```

```
      pop      di;
      pop      si;
      pop      ds;
      pop      es;
      // Set current PSP back to us.
      mov      bx, _psp;
      mov      ah, 0x50;
      int      21h;
   }

   _dos_setvect(i, NULL);
   tsr_print_string("TSR unloaded succesfully.",1);
}

static void print_char(char c)
{
   _asm {
      mov ah, 0eh;
      mov al, byte ptr c;
      int 10h;
   }
}

void tsr_print_string(char far* str, int add_newline)
{
   while (*str) print_char(*str++);
   if (add_newline)
   {
      print_char('¥n');
      print_char('¥r');
   }
}
```

Sourcefile: Unixstat.h

```
#ifndef __UNIXSTAT_H
#define __UNIXSTAT_H

/*
 * copied strait from the linux source
 */


#define S_IFMT  00170000
#define S_IFSOCK 0140000
#define S_IFLNK    0120000
#define S_IFREG  0100000
#define S_IFBLK  0060000
```

```
#define S_IFDIR  0040000
#define S_IFCHR  0020000
#define S_IFIFO  0010000
#define S_ISUID  0004000
#define S_ISGID  0002000
#define S_ISVTX  0001000

#define S_ISLNK(m)  (((m) & S_IFMT) == S_IFLNK)
#define S_ISREG(m)  (((m) & S_IFMT) == S_IFREG)
#define S_ISDIR(m)  (((m) & S_IFMT) == S_IFDIR)
#define S_ISCHR(m)  (((m) & S_IFMT) == S_IFCHR)
#define S_ISBLK(m)  (((m) & S_IFMT) == S_IFBLK)
#define S_ISFIFO(m) (((m) & S_IFMT) == S_IFIFO)
#define S_ISSOCK(m) (((m) & S_IFMT) == S_IFSOCK)

#define S_IRWXU 00700
#define S_IRUSR 00400
#define S_IWUSR 00200
#define S_IXUSR 00100

#define S_IRWXG 00070
#define S_IRGRP 00040
#define S_IWGRP 00020
#define S_IXGRP 00010

#define S_IRWXO 00007
#define S_IROTH 00004
#define S_IWOTH 00002
#define S_IXOTH 00001

#endif
```

Sourcefile: Vfs.h

```
#ifndef __VFS_H
#define __VFS_H

#include "redir.h"


/*
 * These are called by the redirector
 */

int         vfs_setuppartition(int *disk,int *partition,int maxcache,  int Unique);
void        vfs_savecontext();
void        vfs_restorecontext();
void        vfs_shutdown(unsigned l_unit);
```

```
int         vfs_initsearch(unsigned l_unit, char far * path);
int         vfs_findnext(unsigned l_unit, SRCHREC_PTR srchrec, DIRREC_PTR dirrec);
void        vfs_fill_sft(unsigned l_unit, SFTREC_PTR p, DIRREC_PTR dirrec, char far
*fcbname);
unsigned vfs_readfile(unsigned l_unit, SFTREC_PTR p, unsigned size, char far *dst);

unsigned    vfs_secpercluster(unsigned l_unit);
unsigned    vfs_clusters(unsigned l_unit);
unsigned    vfs_freeclusters(unsigned l_unit);
unsigned    vfs_sectorsize(unsigned l_unit);


#endif      // __REDIREX2_H
```

Sourcefile: Vfs.c

```
#pragma check_stack(off)
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#ifdef __BORLANDC__
#include <alloc.h>
#else
#include <malloc.h>
#endif

#include "unixstat.h"
#include "debug.h"
#include "ext2_fs.h"
#include "ems.h"
#include "emscache.h"
#include "diskdev.h"
#include "redir.h"
#include "vfs.h"


/*
 * This module is supposed to be a sort of Virtual File System.
 * It hides the actual file system from the redirector, in our case
 * it's Linux' ext2fs.
 *
 * The 'only' routine that shouldn't be here is ext2_entrylookup.
 *
 * PVS: this interface should definitly get cleaned up. It must provide
 * access to multiple file systems and must support different types
 */
```

```
typedef struct
{
   char             srch_localname[128];
   char             srch_filename[11];
   unsigned long  srch_filesize;
   unsigned long  srch_filetime;
   unsigned long  srch_dir_ino;
   unsigned long  srch_entry_ino;
   long             srch_cur_block;
   long             srch_block_pos;
   unsigned       srch_offset;
   unsigned      srch_i_mode;
   int              n_phantom;
} srch_block_t;

srch_block_t        ffirst_search;
srch_block_t        origfiles_search;

char                *phantom_name="$FILES  ORG";

ldev_t              *Our_ldev;

#define SF_CHARDEV  1
#define SF_BLOCKDEV 2
#define SF_SYMLINK  3
#define SF_SOCKET   4
#define SF_PIPE        5
#define SF_PHANTOM  6

#define PHANTOM_FRONTFORMAT 26

static char      *special_files[]=
              {
                 "",
                 "This is a character device.",
                 "This is a block device.",
                 "This is a symbolic link.",
                 "This is a socket.",
                 "This is a named pipe.",
              };

#define ENTRY_DIR   1
#define ENTRY_FILE  2
#define ENTRY_BOTH  3

#define HIGH_WORD(ul)    ((unsigned) (((ul) & 0xFFFF0000ul ) >> 8))
#define LOW_WORD(ul)     ((unsigned) ((ul) & 0x0000FFFFul))
#define MK_LWORD(u1,u2) ((((unsigned long) (u1)) << 8) | ((unsigned long) (u2)))
```

```
int     Digits_To_Make_Unique=1;
char    bad_chars[]="*?<>¥¥$+=;";


static void convert_filename(char far *filename, char far *newfilename, int offset)
{
   char far    *pc;
   char        c;
   int         i,has_to_chop,n_move;

   _fmemset(newfilename,' ',11);
   pc=newfilename;

/*
 * convert hidden files (starting with a dot) to underscore, unless
 * it's the . or .. entry
 */
   if (filename[0]=='.')
   {
      newfilename[0]='.';
      if (filename[1]==0)
         return;
      else
         if (filename[1]=='.' && filename[2]==0)
         {
            newfilename[1]='.';
            return;
         }
         else
            filename[0]='_';
   }
/*
 * copy filename part
 */
   for (i=0 ; i<8 ; i++)
   {
      c=*filename++;
      if (c=='.' || !c)
         break;
      if (strchr(bad_chars,c))
         c='_';
      *pc++=toupper(c);
   }
   has_to_chop= (i==8 && (*filename!='.' && *filename!=0));


/*
 * search dot for extension of the filename
 */
```

```
   for(; c && c!='.' ; c=*filename++)
      ;

/*
 * copy extension
 */

   pc=newfilename+8;
   for (i=0 ; i<3 && c ; i++)
   {
      c=*filename++;
      if (!c)
         break;
      if (strchr(bad_chars,c) || c=='.')
         c='_';
      *pc++=toupper(c);
   }
/*
 * if either the filename or the extention part is too long,
 * chop it off
 */
   if (has_to_chop || c)
   {
      pc=newfilename+8;
      n_move=i+Digits_To_Make_Unique-1;
      pc+= (n_move>2)? 2 : n_move;
      for (i= 0 ; i<Digits_To_Make_Unique ; i++)
      {
         *pc--= offset % 10 + '0';
         offset/=10;
      }
   }
}

/*
 * This comes strait from the Linux kernel source. I took it from
 * msdos filesystem. Donno how it works, but it does :-)
 */

/* Linear day numbers of the respective 1sts in non-leap years. */
static long day_n[] = { 0,31,59,90,120,151,181,212,243,273,304,334,0,0,0,0 };
       /* JanFebMarApr May Jun Jul Aug Sep Oct Nov Dec */

static uint32 date_unix2dos(uint32 unix_date)
{
   long day,year,nl_day,month;
   unsigned long time,date;

   time = (unix_date % 60)/2+(((unix_date/60) % 60) << 5)+
      (((unix_date/3600) % 24) << 11);
```

```
   day = unix_date/86400l-3652;
   year = day/365;
   if ((year+3)/4+365*year > day) year--;
   day -= (year+3)/4+365*year;
   if (day == 59 && !(year & 3)) {
      nl_day = day;
      month = 2;
   }
   else {
      nl_day = (year & 3) || day <= 59 ? day : day-1;
      for (month = 0; month < 12; month++)
         if (day_n[month] > nl_day) break;
   }
   date = nl_day-day_n[month-1]+1+(month << 5)+(year << 9);

   return (date<<16) | time;
}

static int match_names(char far *entry_name, char far *masked_name)
{
   char  far *pc1=masked_name,far *pc2=entry_name;
   int   i;

   for (i=0 ; i<11 ; i++,masked_name++,entry_name++)
   {
      if ( *masked_name != '?' && *masked_name!=*entry_name)
         break;
   }
   dprintf(D_VFS,"match_names:%-11fs,%-11fs,
returns:%s¥n",pc1,pc2,(i==11)?"True":"False",i);
   return (i==11);
}

static int vfs_isphantomfile(unsigned l_unit, char far *filename)
{
   return match_names(filename,phantom_name);
}

/*
 * Ok, this is the basic part that lookups an entry in an directory
 * (specified by cur_ino). It implements the ffirst/fnext mechanism
 * that is used by DOS. The goto in the middle of the code is needed
 * to gracefully return without touching the state we're in.
 */

char *search_type[]={"ALL","DIR","FILE"};

static unsigned long ext2_entrylookup(unsigned l_unit, srch_block_t *search, char far
*entryname,int what)
{
```

```
   ext2_dir_entry far  *dirent;
   ext2_inode       far   *inode,far *entry_inode;
   cache_entry_t       *e1,*e2,*e3;
   long                   block,bsize=BLOCKSIZE(Our_ldev);
   int                    is_dir;

   search->srch_entry_ino=0l;
   dprintf(D_VFS,"Looking for %s-entries:%-11fs in dir with ino:%lu¥n", search_type[what-
1],entryname,search->srch_dir_ino);
   if (!(e1=ext2_readinode(Our_ldev,l_unit,search->srch_dir_ino)))
      return 0l;
   inode=(ext2_inode far*) e1->data;
   dprintf(D_VFS,"ino_size:%lu, ino_blocks:%lu¥n", inode->i_size,inode-> i_blocks);

   while(search->srch_cur_block*bsize< (long) inode->i_size && !search-> srch_entry_ino)
   {
      block=ext2_bmap(Our_ldev,l_unit,inode,search->srch_cur_block);
      if (!(e2=ext2_readblock(Our_ldev,l_unit,block)))
         goto end_entrylookup;
      dirent=(ext2_dir_entry far *) ((char far *) e2->data+search-> srch_block_pos);
      while (search->srch_block_pos!=bsize)
      {
         _fstrncpy(search->srch_localname,dirent->name,dirent->name_len);
         search->srch_localname[dirent->name_len]=0;
         convert_filename(search->srch_localname,search-> srch_filename,search-
>srch_offset);

         search->srch_block_pos+=dirent->rec_len;
         search->srch_offset++;
         if(match_names(search->srch_filename,entryname))
         {
            if (!(e3=ext2_readinode(Our_ldev,l_unit,dirent->inode)))
            {
               dirent=(ext2_dir_entry far *) (((char far *) dirent) + dirent->rec_len);
               continue;     //skip this entry
            }

            entry_inode=(ext2_inode far*) e3->data;
            is_dir=  S_ISDIR(entry_inode->i_mode) ? 1 : 0;
            search->srch_i_mode=entry_inode->i_mode;
            search->srch_filesize=entry_inode->i_size;
            search->srch_filetime=entry_inode->i_atime;
            dprintf(D_VFS,"Entry-ino:%lu, isdir:%s¥n",dirent->inode, is_dir ? "Yes":"No");
            ext2_releaseinode(Our_ldev,e3);

            if (  what==ENTRY_BOTH ||
                  (what==ENTRY_FILE && !is_dir) ||
                  (what==ENTRY_DIR && is_dir))
            {
               search->srch_entry_ino=dirent->inode;
```

```
                ext2_releaseblock(Our_ldev,e2);
                goto end_entrylookup;
             }
          }
          dirent=(ext2_dir_entry far *) (((char far *) dirent) + dirent->rec_len);
       }
       search->srch_block_pos=0l;
       ext2_releaseblock(Our_ldev,e2);
       search->srch_cur_block++;
    }
end_entrylookup:
    ext2_releaseinode(Our_ldev,e1);
    return search->srch_entry_ino;
}


void vfs_dirlookup(unsigned l_unit, char far *dirname)
{
    char        far *pc,far *pc_ext,far *subdir;
    int         len;
    char        local[11];


    ffirst_search.srch_dir_ino=EXT2_ROOT_INO;
    subdir=dirname+1;
    while((pc=_fstrchr(subdir,'¥¥')) && ffirst_search.srch_dir_ino)
    {
       *pc=0;
       ffirst_search.srch_cur_block=ffirst_search.srch_block_pos=0l;
       ffirst_search.srch_offset=ffirst_search.srch_i_mode=0;
       memset(local,' ',11);
       if ((pc_ext=_fstrchr(subdir,'.')))
       {
          len=pc_ext-subdir;
          _fmemcpy(local,subdir,len);
          _fmemcpy(local+8,subdir+len+1,pc-subdir-len-1);
       }
       else
          _fmemcpy(local,subdir,pc-subdir);
       ffirst_search.srch_dir_ino=ext2_entrylookup(l_unit,&ffirst_search,(char far
*)local,ENTRY_DIR);
       subdir=pc+1;
       *pc='¥¥';
    }
    dprintf(D_VFS,"dirlookup on:%fs,found inode:%lu¥n", dirname,
ffirst_search.srch_dir_ino);
}

/*
 * Check if disk and partition make up a valid device with the right type
```

```
 * if disk==-1, search for the first such a partition. Depending on
 * maxCache we initiale the ext2fs with maximum or minimum cache
 */

int vfs_setuppartition(int *disk, int *partition, int maxcache, int Unique)
{
   int         succes,d,p;

   d=*disk;
   p=*partition;
   Digits_To_Make_Unique=(Unique<1) ? 1: (Unique > 3) ? 3: Unique;

   if (d==-1)
   {
      dprintf(D_ALWAYS,"Searching for an ext2 file system...¥n");
      for (d=0 ; d< MAX_HD ; d++)
         for (p=0 ; p<MAX_LDEV ; p++)
         {
            Our_ldev=&lDevices[d][p];
            if (Our_ldev->type==131)
            {
               dprintf(D_ALWAYS,"Found one on hd%c%c¥n",d+'a',p+'0');
               goto got_partition;
            }
         }
      dprintf(D_ALWAYS,"There aren't any!¥n");
      return 0;
   }
   else
      Our_ldev=&lDevices[d][p];

got_partition:
   if (Our_ldev->type==131)
   {
      dprintf(D_EXT2,"Found ext2 file system on partition hd%c%c¥n", d+'a',p+'0');
      if(maxcache)
         succes=ext2_init(Our_ldev, 2, 48);
      else
         succes=ext2_init(Our_ldev, 1, 1);
      if (!succes)
         dprintf(D_ALWAYS,"Ouch, could not setup ext2 file system!¥n");
   }
   else
      dprintf(D_ALWAYS,"Could not find an ext2 file system on hd%c%c¥n", d+'a',p+'0');

   *disk=d;
   *partition=p;
   return succes;
}
```

```
char old_path[128]="";
unsigned long old_ino;

int vfs_initsearch(unsigned l_unit, char far * path)
{
   char     far *pc;

   if ((pc=_fstrrchr(path+1,'¥¥')))
   {
      *pc=0;
      if (_fstrcmp(path,old_path)==0)
      {
         ffirst_search.srch_dir_ino=old_ino;
      }
      else
      {
         *pc='¥¥';
         vfs_dirlookup(l_unit, path);    // sets ffirst_search.srch_dir_ino
         *pc=0;
         _fstrcpy(old_path,path);
         old_ino=ffirst_search.srch_dir_ino;
      }
      *pc='¥¥';
   }
   else
      ffirst_search.srch_dir_ino=EXT2_ROOT_INO;

   ffirst_search.srch_cur_block=ffirst_search.srch_block_pos=0l;
   ffirst_search.srch_offset=ffirst_search.srch_i_mode=0;
   ffirst_search.n_phantom=0;
   dprintf(D_VFS,"initsearch:%fs, found inode:%lu¥n", path,ffirst_search.srch_dir_ino);
   return (int) ffirst_search.srch_dir_ino;
}

int vfs_findnext(unsigned l_unit, SRCHREC_PTR srchrec, DIRREC_PTR dirrec)
{
   int   what=(srchrec->attr_mask & ATTR_SUBDIR) ? ENTRY_BOTH : ENTRY_FILE;


   if (ffirst_search.n_phantom)
      return 0;

   if (ext2_entrylookup(l_unit,&ffirst_search, srchrec->srch_mask,what))
   {
      dprintf(D_VFS,"ffnext: found entry:%-11s¥n", ffirst_search.srch_filename);
      _fmemcpy(dirrec->file_name,ffirst_search.srch_filename,11);
      dirrec->file_attr=S_ISDIR(ffirst_search.srch_i_mode)? ATTR_SUBDIR : ATTR_READONLY;
      dirrec->start_sector=(uint16) ffirst_search.srch_entry_ino;
      dirrec->file_size=S_ISDIR(ffirst_search.srch_i_mode)? 0l: (int32)
ffirst_search.srch_filesize;
```

```
      dirrec->file_time=date_unix2dos(ffirst_search.srch_filetime);
      srchrec->dir_entry_no=ffirst_search.srch_offset;
      srchrec->dir_sector=(uint16) ffirst_search.srch_dir_ino;
   }
   else
   {
      ffirst_search.n_phantom++;
      if (!match_names(phantom_name,srchrec->srch_mask))
         return 0;

      dprintf(D_VFS,"ffnext: made phantom file:%s¥n",phantom_name);
      _fmemcpy(dirrec->file_name,phantom_name,11);
      dirrec->file_attr=ATTR_READONLY;
      dirrec->start_sector=0;
      dirrec->file_size=0;
      dirrec->file_time=date_unix2dos(ffirst_search.srch_filetime);
      srchrec->dir_entry_no=0;
      srchrec->dir_sector=0;
   }

   return 1;
}

/*
 * Fill in the entry in the System File Table. If it's a special file
 * we'll put some different information in this entry (f.e. a pointer
 * to a description string).
 */

void vfs_fill_sft(unsigned l_unit, SFTREC_PTR p, DIRREC_PTR dirrec, char far *fcbname)
{
    _fmemcpy(p->file_name, fcbname, 11);
    p->file_attr=dirrec->file_attr;
    p->file_time=dirrec->file_time;
    p->rel_sector=HIGH_WORD(ffirst_search.srch_entry_ino);
    p->abs_sector=LOW_WORD(ffirst_search.srch_entry_ino);

   p->dir_sector=   S_ISCHR(ffirst_search.srch_i_mode) ? SF_CHARDEV :
               S_ISBLK(ffirst_search.srch_i_mode) ? SF_BLOCKDEV :
               S_ISLNK(ffirst_search.srch_i_mode) ? SF_SYMLINK :
               S_ISSOCK(ffirst_search.srch_i_mode) ? SF_SOCKET :
               S_ISFIFO(ffirst_search.srch_i_mode) ? SF_PIPE :
               match_names(fcbname,phantom_name)? SF_PHANTOM : 0;

   if (p->dir_sector == SF_PHANTOM)
   {
      origfiles_search.srch_dir_ino=ffirst_search.srch_dir_ino;
      origfiles_search.srch_cur_block=0;
      origfiles_search.srch_block_pos=0;
      origfiles_search.srch_offset=0;
```

```
      p->start_sector=0;
      p->file_size= (ffirst_search.srch_cur_block+1) * BLOCKSIZE(Our_ldev) +
PHANTOM_FRONTFORMAT * ffirst_search.srch_offset;
   }
   else
      if (p->dir_sector)
      {
         p->file_size=strlen(special_files[p->dir_sector]);
         p->start_sector=(unsigned) special_files[p->dir_sector];
      }
      else
      {
            p->dir_sector=0;
            p->file_size=dirrec->file_size;
            p->start_sector=dirrec->start_sector;
      }
    p->dir_entry_no=(uint8) ffirst_search.srch_offset;
}


/*
 * The $file.org lists for each directory entry, the type, the access
 * list, the converted dos name and the original name.
 * example:
 *
 * drwxr-xr-x LOST_FOU 2 = lost+found
 */

static char format_buffer[PHANTOM_FRONTFORMAT+1];

static char *format_phantomline(srch_block_t *search)
{
   int      i;
   uint16   mode= (uint16) search->srch_i_mode;

   format_buffer[0] =  S_ISCHR(search->srch_i_mode) ? 'c' :
            S_ISBLK(search->srch_i_mode) ? 'b' :
            S_ISLNK(search->srch_i_mode) ? 'l' :
            S_ISSOCK(search->srch_i_mode) ? 's' :
            S_ISFIFO(search->srch_i_mode) ? 'p' :
            S_ISDIR(search->srch_i_mode) ? 'd' : ' ';

   for (i=0 ; i<9 ; i++)
   {
      format_buffer[9-i]= (mode & 1) ? "xrw"[i%3] : '-';
      mode>>=1;
   }
   format_buffer[10]=' ';
   _fmemcpy(&format_buffer[11],search->srch_filename,8);
   format_buffer[19]=' ';
```

```
   _fmemcpy(&format_buffer[20],search->srch_filename+8,3);
   format_buffer[23]=' ';
   format_buffer[24]='=';
   format_buffer[25]=' ';

   return format_buffer;
}


static unsigned vfs_readphantomfile(SFTREC_PTR p, unsigned size, char far *dst)
{
   char             *src;
   unsigned         to_go,n,prev_offset;
   unsigned long  ino;

   prev_offset=p->start_sector;
   to_go=size;

   dprintf(D_VFS,"readphantom: read size:%u¥n",size);

   if (prev_offset==0)
   {
      ino=ext2_entrylookup(0, &origfiles_search, "???????????", ENTRY_BOTH);
      if (!ino)
      {
         dprintf(D_VFS,"lookup returned inode:%lu¥n",ino);
         p->file_size=p->file_pos;
         return 0;
      }
   }



   do
   {
      dprintf(D_VFS,"readphantom: %s item: %s¥n", prev_offset ? "finishing" : "adding",
origfiles_search.srch_filename);
      if (prev_offset < PHANTOM_FRONTFORMAT )
      {
         src=format_phantomline(&origfiles_search);
         n=PHANTOM_FRONTFORMAT-prev_offset;
         n=(to_go < n) ? to_go : n;
         p->start_sector=n;
         _fmemcpy(dst,src+prev_offset,n);
         to_go-=n;
         prev_offset+=n;
         dst+=n;
      }
      if (to_go)
      {
         n=_fstrlen(origfiles_search.srch_localname+prev_offset-PHANTOM_FRONTFORMAT);
```

```
         n=(to_go<n) ? to_go : n;
         p->start_sector+= n;

         _fmemcpy(dst,origfiles_search.srch_localname+prev_offset-PHANTOM_FRONTFORMAT,n);
         to_go-=n;
         dst+=n;
      }
      if (to_go)
      {
         *dst++=10;
         p->start_sector=prev_offset=0;
         to_go--;
      }


   } while (to_go && ext2_entrylookup(0, &origfiles_search, "???????????", ENTRY_BOTH));

   if (to_go)
   {
      p->file_size+=p->file_pos+size-to_go;
   }

   p->file_pos+=size-to_go;
   return size-to_go;
}


static unsigned vfs_readspecialfile(SFTREC_PTR p, unsigned size, char far *dst)
{
   char             *src;
   unsigned         n,file_pos;
   unsigned long cur_ino;


   cur_ino=MK_LWORD(p->rel_sector,p->abs_sector);
   file_pos=(unsigned) p->file_pos;
   src = ((char *) p->start_sector) + file_pos;
   n=((unsigned) p->file_size) - file_pos;

   dprintf(D_VFS,"ext2_readspecialfile (%u):%11fs, ino:%ul, fp:%u;, size:%u¥n", p-
>dir_sector,p->file_name,cur_ino,p->file_pos,size);

   n=(size < n) ? size : n;
   _fmemcpy(dst,src,n);
   p->file_pos+=n;
   return n;
}


unsigned vfs_readfile(unsigned l_unit, SFTREC_PTR p, unsigned size, char far *dst)
```

```
{
   cache_entry_t *e1,*e2;
   ext2_inode       far *inode;
   unsigned         n,to_go;
   unsigned long cur_ino, block, lblock,bsize=BLOCKSIZE(Our_ldev),offset;
   char             far *src;

   if(p->dir_sector==SF_PHANTOM)
      return vfs_readphantomfile(p,size,dst);

   if(p->dir_sector)
      return vfs_readspecialfile(p,size,dst);

   cur_ino=MK_LWORD(p->rel_sector,p->abs_sector);
   lblock=p->file_pos / bsize;
   offset=p->file_pos % bsize;
   to_go=size;

   dprintf(D_VFS,"ext2_readfile:%11fs, ino:%ul, fp:%u;, size:%u¥n",p-> file_name,
cur_ino,p->file_pos,size);


   if (!(e1=ext2_readinode(Our_ldev,l_unit,cur_ino)))
   {
      dprintf(D_ALWAYS,"ext_readfile: could not read inode:%lu¥n,", cur_ino);
      goto vfs_readfile_end;
   }
   inode=(ext2_inode far*) e1->data;

   for(to_go=size; to_go ; lblock++)
   {
      block=ext2_bmap(Our_ldev,l_unit,inode,lblock);
      if (!(e2=ext2_readblock(Our_ldev,l_unit,block)))
      {
         dprintf(D_ALWAYS,"ext_readfile: readblock %lu failed¥n",lblock);
         dprintf(D_ALWAYS,"filepos:%lu, filesz:%lu, size:%u, togo:%u¥n",p->file_pos,p-
>file_size,size,to_go);
         p->file_size=p->file_pos+size-to_go;
                              // make sure next read is EOF !
         break;
      }
      src=((char far *) e2->data)+offset;
      n=(unsigned) (bsize-offset);
      n=(n>to_go) ? to_go : n;
      _fmemcpy(dst,src,n);
      ext2_releaseblock(Our_ldev,e2);
      offset=0l;
      to_go-=n;
      dst+=n;
   }
```

```
   ext2_releaseinode(Our_ldev,e1);
vfs_readfile_end:
   p->file_pos+=size-to_go;
   return size-to_go;
}


void vfs_shutdown(unsigned l_unit)
{
   ext2_shutdown(Our_ldev);
}


void vfs_savecontext()
{
   ext2_savecontext(Our_ldev);
}


void vfs_restorecontext()
{
   ext2_restorecontext(Our_ldev);
}


unsigned vfs_secpercluster(unsigned l_unit)
{
   return Our_ldev->sec_perblock;
}


unsigned vfs_clusters(unsigned l_unit)
{
   return (unsigned) (ext2_blockscount(Our_ldev) / Our_ldev-> sec_perblock);
}


unsigned vfs_freeclusters(unsigned l_unit)
{
   return (unsigned) (ext2_freeblockscount(Our_ldev) / Our_ldev->sec_perblock);
}


unsigned vfs_sectorsize(unsigned l_unit)
{
   return BYTES_PERSECTOR;
}
```

Sourcefile: Vsprintf.h

```
#ifndef __VSPRINTF_H
#define __VSPRINTF_H

#include <stdarg.h>

int simple_atoi(char *);
int simple_vsprintf(char *buf, const char *fmt, va_list args);
int simple_sprintf(char * buf, const char *fmt, ...);

#endif          // __VSPRINTF_H
```

Sourcefile: Vsprintf.c

```
#pragma check_stack(off)
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include "vsprintf.h"



unsigned _fstrnlen(const char far * s, unsigned count)
{
   const char far *sc;

   for (sc = s; *sc != '¥0' && count--; ++sc)
      /* nothing */;
   return sc - s;
}

unsigned long simple_strtoul(const char *cp,char **endp,unsigned int base)
{
   unsigned long result = 0,value;

   if (!base) {
      base = 10;
      if (*cp == '0') {
         base = 8;
         cp++;
         if ((*cp == 'x') && isxdigit(cp[1])) {
            cp++;
            base = 16;
```

```
        }
      }
   }
   while (isxdigit(*cp) && (value = isdigit(*cp) ? *cp-'0' : (islower(*cp)
      ? toupper(*cp) : *cp)-'A'+10) < base) {
      result = result*base + value;
      cp++;
   }
   if (endp)
      *endp = (char *)cp;
   return result;
}

int simple_atoi(char *str)
{
   return (int) simple_strtoul(str,0,10);
}

/* we use this so that we can do without the ctype library */
#define is_digit(c) ((c) >= '0' && (c) <= '9')

static int skip_atoi(const char **s)
{
   int i=0;

   while (is_digit(**s))
      i = i*10 + *((*s)++) - '0';
   return i;
}

#define ZEROPAD  1     /* pad with zero */
#define SIGN  2        /* unsigned/signed long */
#define PLUS  4        /* show plus */
#define SPACE 8     /* space if plus */
#define LEFT  16       /* left justified */
#define SPECIAL  32    /* 0x */
#define LARGE 64    /* use 'ABCDEF' instead of 'abcdef' */


static char * number(char * str, long num, int base, int size, int precision, int type)
{
   char c,sign,tmp[66];
   const char *digits="0123456789abcdefghijklmnopqrstuvwxyz";
   int res,i;

   if (type & LARGE)
      digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   if (type & LEFT)
      type &= ~ZEROPAD;
   if (base < 2 || base > 36)
```

```
   return 0;
c = (type & ZEROPAD) ? '0' : ' ';
sign = 0;
if (type & SIGN) {
   if (num < 0) {
      sign = '-';
      num = -num;
      size--;
   } else if (type & PLUS) {
      sign = '+';
      size--;
   } else if (type & SPACE) {
      sign = ' ';
      size--;
   }
}
if (type & SPECIAL) {
   if (base == 16)
      size -= 2;
   else if (base == 8)
      size--;
}
i = 0;
if (num == 0)
   tmp[i++]='0';
else while (num != 0)
{
   res = (int) ((unsigned long) num) % (unsigned) base;
   num = ((unsigned long) num) / (unsigned) base;
   tmp[i++] = digits[res];
}
if (i > precision)
   precision = i;
size -= precision;
if (!(type&(ZEROPAD+LEFT)))
   while(size-->0)
      *str++ = ' ';
if (sign)
   *str++ = sign;
if (type & SPECIAL)
   if (base==8)
      *str++ = '0';
   else if (base==16) {
      *str++ = '0';
      *str++ = digits[33];
   }
if (!(type & LEFT))
   while (size-- > 0)
      *str++ = c;
while (i < precision--)
```

```
      *str++ = '0';
   while (i-- > 0)
      *str++ = tmp[i];
   while (size-- > 0)
      *str++ = ' ';
   return str;
}

int simple_vsprintf(char *buf, const char *fmt, va_list args)
{
   int              len;
   unsigned long num;
   int              i, base;
   char *           str;
   char far *       s;

   int flags;    /* flags to number() */

   int field_width; /* width of output field */
   int precision;   /* min. # of digits for integers; max
               number of chars for from string */
   int qualifier;   /* 'h', 'l', or 'L' for integer fields */

   for (str=buf ; *fmt ; ++fmt) {
      if (*fmt != '%') {
         *str++ = *fmt;
         continue;
      }

      /* process flags */
      flags = 0;
      repeat:
         ++fmt;      /* this also skips first '%' */
         switch (*fmt) {
            case '-': flags |= LEFT; goto repeat;
            case '+': flags |= PLUS; goto repeat;
            case ' ': flags |= SPACE; goto repeat;
            case '#': flags |= SPECIAL; goto repeat;
            case '0': flags |= ZEROPAD; goto repeat;
            }

      /* get field width */
      field_width = -1;
      if (is_digit(*fmt))
         field_width = skip_atoi(&fmt);
      else if (*fmt == '*') {
         ++fmt;
         /* it's the next argument */
         field_width = va_arg(args, int);
         if (field_width < 0) {
```

```
            field_width = -field_width;
            flags |= LEFT;
        }
    }

    /* get the precision */
    precision = -1;
    if (*fmt == '.') {
        ++fmt;
        if (is_digit(*fmt))
            precision = skip_atoi(&fmt);
        else if (*fmt == '*') {
            ++fmt;
            /* it's the next argument */
            precision = va_arg(args, int);
        }
        if (precision < 0)
            precision = 0;
    }

    /* get the conversion qualifier */
    qualifier = -1;
    if (*fmt == 'h' || *fmt == 'l' || *fmt == 'L' || *fmt == 'f') {
        qualifier = *fmt;
        ++fmt;
    }

    /* default base */
    base = 10;

    switch (*fmt) {
    case 'c':
        if (!(flags & LEFT))
            while (--field_width > 0)
                *str++ = ' ';
        *str++ = (unsigned char) va_arg(args, int);
        while (--field_width > 0)
            *str++ = ' ';
        continue;

    case 's':
        if (qualifier=='f')
            s = (char far *) va_arg(args, char far *);
        else
            s = (char far *) va_arg(args, char *);

        if (!s)
            s = "<NULL>";

        //len = _fstrnlen(s, precision);
```

```
        len = _fstrnlen(s, field_width);

        if (!(flags & LEFT))
           while (len < field_width--)
              *str++ = ' ';
        for (i = 0; i < len; ++i)
           *str++ = *s++;
        while (len < field_width--)
           *str++ = ' ';
        continue;

     case 'p':
        if (field_width == -1) {
           field_width = 2*sizeof(void *);
           flags |= ZEROPAD;
        }
        str = number(str, (unsigned long) va_arg(args, void *), 16,
           field_width, precision, flags);
        continue;


     case 'n':
        if (qualifier == 'l') {
           long * ip = va_arg(args, long *);
           *ip = (str - buf);
        } else {
           int * ip = va_arg(args, int *);
           *ip = (str - buf);
        }
        continue;

     /* integer number formats - set up the flags and "break" */
     case 'o':
        base = 8;
        break;

     case 'X':
        flags |= LARGE;
     case 'x':
        base = 16;
        break;

     case 'd':
     case 'i':
        flags |= SIGN;
     case 'u':
        break;

     default:
        if (*fmt != '%')
```

```
            *str++ = '%';
         if (*fmt)
            *str++ = *fmt;
         else
            --fmt;
         continue;
      }
      if (qualifier == 'l')
         num = va_arg(args, unsigned long);
      else if (qualifier == 'h')
         if (flags & SIGN)
            num = va_arg(args, short);
         else
            num = va_arg(args, unsigned short);
      else if (flags & SIGN)
         num = va_arg(args, int);
      else
         num = va_arg(args, unsigned int);
      str = number(str, num, base, field_width, precision, flags);
   }
   *str = '¥0';
   return str-buf;
}

int simple_sprintf(char * buf, const char *fmt, ...)
{
   va_list args;
   int i;

   va_start(args, fmt);
   i=simple_vsprintf(buf,fmt,args);
   va_end(args);
   return i;
}
```