



von Franz-Josef Reichert

Das Amiga-DOS ist eine so wichtige und umfangreiche Komponente, daß wir auf die Beschreibung des DOS mehr Platz verwenden, als für die übrigen Teile. Auch wollen wir die Informationen an einem praktischen Programmbeispiel vertiefen. So sind aus dem Teil 4 — Amiga-DOS — zwei Teile geworden. Freuen Sie sich schon jetzt auf den zweiten Teil, den wir in der Ausgabe 9 des AMIGA-Magazins veröffentlichen.

»DOS« leitet sich von dem Begriff »Disk Operating System« ab, was soviel bedeutet wie »Betriebssystem für Disketten«. Diese Bezeichnung aus

der Urzeit der Computergeschichte hat im Laufe der Zeit für moderne Computer vom Zuschnitt eines Amigas einen gewaltigen Bedeutungswandel vollzogen.

Stand früher noch eindeutig die Diskette im Vordergrund, muß ein heutiges DOS weitaus komplexere Aufgaben bewältigen als nur das Lesen und Beschreiben einer Diskette. Man legt den Begriff daher wesentlich weitläufiger aus und versteht unter einem DOS denjenigen Betriebssystemteil, der die Kommunikation des Rechnersystems mit den Peripheriegeräten über die Schnittstellen steuert. Das DOS beschäftigt sich also hauptsächlich mit Massenspeichern wie Disketten und Festplatten, Eingabegeräten wie der Tastatur und Ausgabegeräten wie Drucker und Bildschirm.

Betriebssystem Amiga-DOS

Das Amiga-DOS ist als Betriebssystemteil erst relativ spät dem Gesamtkonzept hinzugefügt worden. Auch die Bezeichnung »Amiga«-DOS ist eigent-

AMIGA-II

In diesem Kursteil geht es um die Ein- und Aus- vor allem die Diskettenoperationen auf munikation mit den Peripheriegeräten, die und die Ausgabe von Texten auf gaben erledigt für uns das Amiga-DOS mit

lich nicht ganz zutreffend, denn dieses DOS-System ist beileibe nicht für den Amiga entwickelt worden. Das liegt zum einen daran, daß der Amiga ursprünglich ohne Diskettenlaufwerk geplant war und daher keine Notwendigkeit für ein DOS bestand. Der Amiga sollte eine reine Telespielkonsole werden.

Zum anderen ist nach Änderung des Amiga-Konzepts zum PC-ähnlichen Computer kurz vor der Markteinführung die Zeit zur Entwicklung eines eigenen, multitaskingfähigen DOS einfach zu knapp geworden. Es wurde also ein bereits existierendes DOS-System für den Amiga umgeschrieben

(portiert). Metacomco lieferte damals in Rekordzeit eine Anpassung des Systems »TRIP-POS« für den Amiga.

Noch bis heute ist erkennbar, daß sich das Amiga-DOS nicht nahtlos in das System eingefügt hat. Die Hauptursache dafür ist, daß der überwiegende Teil in der Programmiersprache BCPL geschrieben ist, im Gegensatz zu den anderen Betriebssystemteilen, die ausnahmslos in »C« oder Assembler codiert worden sind. Das ist jedoch keineswegs ein Nachteil. Das relativ selten verwendete BCPL ist nämlich eine sehr maschinennahe Sprache und ermöglicht — ähnlich wie

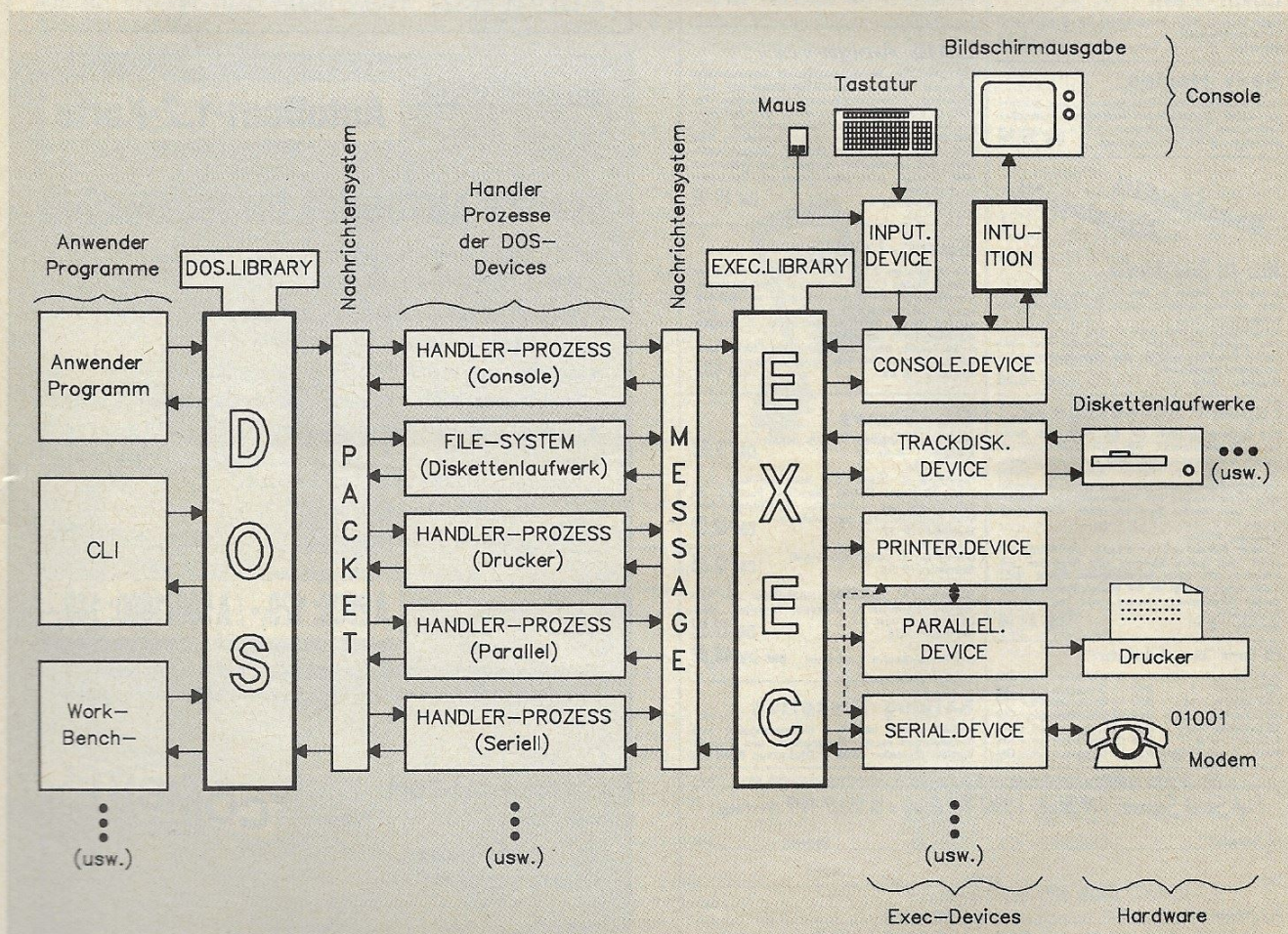


Bild 1. Der Zusammenhang zwischen der DOS-Library und Exec wird deutlich, wenn man diese Grafik betrachtet

Insider

Ausgabe auf dem Amiga. Hierunter fallen auf Dateiebene, aber auch die Kom- Verarbeitung von Tastatureingaben auf dem Bildschirm. Diese Auf- der DOS-Library und den Handlern.

Assembler — ausgesprochen schnelle und kompakte Programme.

Einige Besonderheiten müssen dabei jedoch vom Programmierer beachtet werden. So kennt BCPL in der 68000er-Implementation als kleinstes adressierbares Objekt nur das 32-Bit-Langwort. Alle Strukturen, Strings und andere abgeleitete Datentypen müssen also stets an Langwortgrenzen ausgerichtet sein, das heißt, ihre Adressen müssen ohne Rest durch vier teilbar sein. Soll ein BCPL-Pointer (BPTR) in einen C-Pointer (APTR) konvertiert werden, muß sein Wert mit vier multipliziert werden. Umgekehrt wird ein APTR über die Division durch vier in einen BPTR überführt.

In der Praxis erreicht man dies durch bitweises Verschieben um zwei Stellen. Einer ähnlich ungewohnten Konvention folgen die Strings. Die BCPL-Strings (BSTR) werden nicht durch ein Null-Byte abgeschlossen, wie in anderen Programmiersprachen üblich. Die Länge ist im ersten Byte des Strings abgelegt, darauf folgen erst die Zeichen des Strings. Behält man diese Besonderheiten im Hinterkopf, so gestaltet sich der Umgang mit Amiga-DOS-Strukturen nicht wesentlich komplizierter als der mit den übrigen Systemstrukturen. Die Einbindung des Amiga-DOS ins Gesamtsystem erfolgt wie bei allen anderen Betriebssystemteilen des Amiga über die uns bereits bekannte Library-Struktur.

Amiga-DOS als System

Trotz seiner zentralen Bedeutung kommen viele Amiga-Anwender beim Umgang mit ihrem Computer mit Amiga-DOS scheinbar nicht in Berührung. Mit dem Wort Amiga-DOS verbinden die meisten Amiga-Fans sofort den Begriff »CLI«. Im Gegensatz zur Amiga-typischen Workbench stellen das CLI und

die »Shell« (ab Version 1.3) als alternative Benutzerschnittstelle konventionellen Zuschnitts tatsächlich einen direkteren Bezug zum Amiga-DOS her. Jedoch macht auch die Workbench regen Gebrauch von den DOS-Funktionen, nur eben unmerklich für den Anwender. Über Amiga-DOS kann jedes Programm in komfortabler Weise auf Massenspeicher oder Peripherie zugreifen. Zur Kommunikation mit den »physikalischen« Geräten greift Amiga-DOS aber auf andere Betriebssystemteile zurück. So werden alle Operationen von Amiga-DOS mit den Peripheriegeräten über das bereits besprochene Device-Konzept von Exec abgewickelt (Bild 1). Die Tastatureingaben werden vom »keyboard.device« und »input.device« übernommen, die Bildschirmausgabe erfolgt über das »console.device«.

Dabei wird außerdem intensiver Gebrauch von Intuition gemacht, insbesondere von Windows und Automatik-Requestern. Alle übrigen Geräte wie Drucker, Diskettenlaufwerke und Festplatten werden ebenfalls über das entsprechende Exec-Device angesprochen.

Wer die Peripheriegeräte über das Amiga-DOS anspricht, kann die direkte Kommunikation mit den Exec-Devices elegant umgehen. Dabei muß jedoch beachtet werden, daß in einer Multitasking-Umgebung nicht jeder Task nach Belieben auf verfügbare Geräte zugreifen darf. Es muß ein Weg gefunden werden, nach dem für die konkurrierenden Tasks eine geordnete Zuteilung der Ressourcen erfolgen kann.

Jedes Gerät verfügt über einen symbolischen Namen und einen eigenen »Handler-Prozeß«. Was ein Handler-Prozeß ist, werden wir in einem folgenden Abschnitt noch ausführlicher besprechen. Im Moment soll es uns genügen, zu wissen, daß er für uns die Kommunikation mit einem Exec-

Device übernimmt. Das Anwenderprogramm kommuniziert also nicht direkt mit der Hardware, sondern sendet seine Schreib-/Lese-Befehle an den entsprechenden Handler-Prozeß. Dort werden alle empfangenen Befehle in eine Warteschlange eingereiht und nacheinander ausgeführt, wenn über Exec dem Handler-Prozeß der Zugriff auf das betreffende Gerät erlaubt wird. Das aufrufende Programm braucht also nicht zu warten, bis der Vorgang beendet ist, sondern kann sich gleich der nächsten Aufgabe zuwenden.

Aus der Sicht des Anwenders geht dies so schnell, daß es den Anschein hat, als würde die angeforderte Hardware ausschließlich für den eigenen Task zur Verfügung stehen. Dieses Konzept der »asynchronen« Gerätesteuerung ist

grundlegend für die Leistungsfähigkeit eines Multitaskingsystems. Amiga-DOS-Devices stellen die höchste Ebene dar, auf der der Amiga-Benutzer direkt mit einem Peripheriegerät kommunizieren kann. Die untenstehende Tabelle zeigt einige wichtige DOS-Devices in der Übersicht.

Meistens ist die bloße Angabe des DOS-Devices aber zu ungenau, um die erwünschte Operation durchzuführen. Einen Text können Sie gerade noch ohne weitere Spezifikation an einen Drucker schicken, er wird eben nur ausgedruckt. Es genügt allerdings nicht, zu speichernde Daten einfach an ein Diskettenlaufwerk zu senden. Wir brauchen also zu den dateiorientierten Devices noch weitere Organisationsstrukturen, die uns eine differenziertere Dateiverwaltung ermögli-

DOS-Device ->	Handler ->	Exec-Device
DFO:	File System	trackdisk.device
NEWCON:	Newcon-Handler	console.device
PRT:	Port-Handler	printer.device
SER:	Post-Handler	serial.device
RAM:	RAM-Handler	—
RAD:	File System	ramdrive.device
SPEAK:	Speak-Handler	narrator.device
PIPE:	Pipe-Handler	—

Tabelle. Jedes Device verfügt über einen Handler

Teil 4

KURSÜBERSICHT

Amiga-Insider verdeutlicht die Zusammenhänge und Abläufe im Betriebssystem des Amiga. Vorkenntnisse im Programmieren sind notwendig. Wenn Sie den Kurs verfolgen, machen wir Sie zum Amiga-Insider.

TEIL 1: Übersicht, Betriebssystem-Kern (Exec), Nachrichtenübermittlung, Signale, Speicherverwaltung, Bibliotheken, Ausnahmeforderungen

TEIL 2: Intuition — Die Bedienoberfläche, Intuition-Direktkommunikation, input.device, diskfont.library, icon.library

TEIL 3: Geräte (Devices), input.device, keyboard.device, gameport.device, console.device, timer.device, serial.device, parallel.device, printer.device, audio.device, clipboard.device

TEIL 4: Zugriff auf die Diskette — dos.library, trackdisk.library

TEIL 5: Grafik und Text auf dem Bildschirm — graphics.library, layers.library

TEIL 6: Kochrezept — Wie baut man sich ein eigenes Device oder eine eigene Library?

chen. Das Stichwort lautet »hierarchisches Dateisystem«, das durch einen erweiterten Handler-Prozeß, das »File System«, auf dateiorientierten Devices realisiert wird. Dazu sind einige weitere Amiga-DOS-Objekte erforderlich, die erst eine systematische Gliederung unserer Datenbestände ermöglichen.

Das umfassendste Objekt des Filesystems ist eine Diskette selbst, die mit Namen versehen wird. Im Amiga-DOS wird sie »Volume« genannt, im Gegensatz zum »Device«, welches ja nur das Diskettenlaufwerk als solches bezeichnet. Die nächstniedrigere Stufe ist das »Directory«, also das Dateiverzeichnis, welches wiederum weitere Directories enthalten kann. In dieser Kategorie besteht auch die Möglichkeit, sogenannte »virtuelle Geräte«, auch »Pseudo-Devices« genannt, zu erzeugen. Dies geschieht auf Benutzerebene mit dem CLI-Befehl ASSIGN. Amiga-DOS stellt schon bei seiner Initialisierung einige dieser Geräte zur Verfügung, dies sind beispielsweise Geräte wie FONTS:, LIBS:, SYS: etc. Sie erlauben die direkte Ansprache einer bestimmten Hierarchieebene, ohne den zumeist auf-



wendigen Weg über den Pfadnamen nehmen zu müssen. Die niedrigsten Objekte des Filesystems sind schließlich die »Files«, die Dateien selbst, welche ja erst die eigentlichen Daten enthalten.

Erst mit diesen vier Objekten Device, Volume, Directory und File haben wir die Möglichkeit, unsere Daten nach den von uns gewünschten Kriterien zu verwalten. Diese Objekte werden uns vom Filesystem-Prozeß zur Verfügung gestellt, mit dem wir nur zu kommunizieren brauchen, um Directories oder Files auf einem Volume anzulegen oder Daten zu lesen und zu schreiben. Wie diese Kommunikation genau abläuft, werden wir in einem späteren Abschnitt behandeln.

Wir wollen im folgenden Abschnitt die interne Speicherung der vom Amiga-DOS verwendeten Objekte anhand ihrer Datenstrukturen besprechen und die Bedeutung der Einträge erklären.

Ausgangsbasis für unsere Betrachtungen ist das Verbindungsglied des Amiga-DOS zum übrigen Betriebssystem, die Struktur der DOS-Library (Bild 2). Sie erhalten die Adresse dieser Struktur durch Aufruf der Exec-Funktion »OpenLibra-

ry()«. Hier soll uns nur der Querverweis »dl_Root« zur Struktur »RootNode« interessieren. Der RootNode enthält einen Verweiseintrag namens »rn_Info«, der auf die »DOSInfo« Struktur zeigt. Beachten Sie, daß es sich hierbei um einen BPTR handelt! Die absolute Adresse der DOS-Info-Struktur erhalten Sie also, indem Sie rn_Info mit vier multiplizieren. Die DOSInfo-Struktur enthält einen BPTR auf die erste »DeviceNode« Struktur, »di_DevInfo«.

Sie finden in den Include-Files zum Amiga-DOS diese Struktur einmal als »DeviceNode« und einmal als »DeviceList« beschrieben. Lassen Sie sich nicht verunsichern, es handelt sich dabei stets um genau dieselbe Struktur, nur kann diese je nach ihrem Typ unter-

schiedliche Werte enthalten. Welche Struktur Sie gerade vor sich haben, können Sie am Feld »dn_Type« erkennen.

Hier sind die Werte DLT_DEVICE (0), DLT_DIRECTORY (1) oder DLT_VOLUME (2) erlaubt. Demnach handelt es sich entweder um eine DeviceNode- oder eine DeviceList-Struktur. Die erste kann dabei den Typ DLT_DEVICE wie auch den Typ DLT_DIRECTORY beinhalten, die andere ist dagegen aus-

also ein Directory, das wie ein Device angesprochen werden kann.

Wir wollen zur Vereinfachung stets von der DeviceNode-Struktur sprechen, auch wenn diese in Wirklichkeit die andere Struktur ist. Über den Eintrag »dn_Next« sind alle Device Nodes in einer einfachen Liste verkettet. »dn_Next« der letzten DeviceNode zeigt auf Null. Den Namen der DeviceNode-Struktur finden Sie im Feld »dn_Name«. Alle übrigen Einträge werden vom Amiga-DOS intern zur Kommunikation mit den Handler-Prozessen benutzt.

Eine interessante Eigenschaft des Eintrags »dn_Startup« sei hier noch erwähnt, die es Ihnen ermöglicht, über eine DeviceNode vom Typ

DeviceNode-Struktur

schließlich dem Typ DLT_VOLUME vorbehalten. Dabei bezeichnet der Typ DLT_DIRECTORY ein »pseudo-Device« wie oben beschrieben,

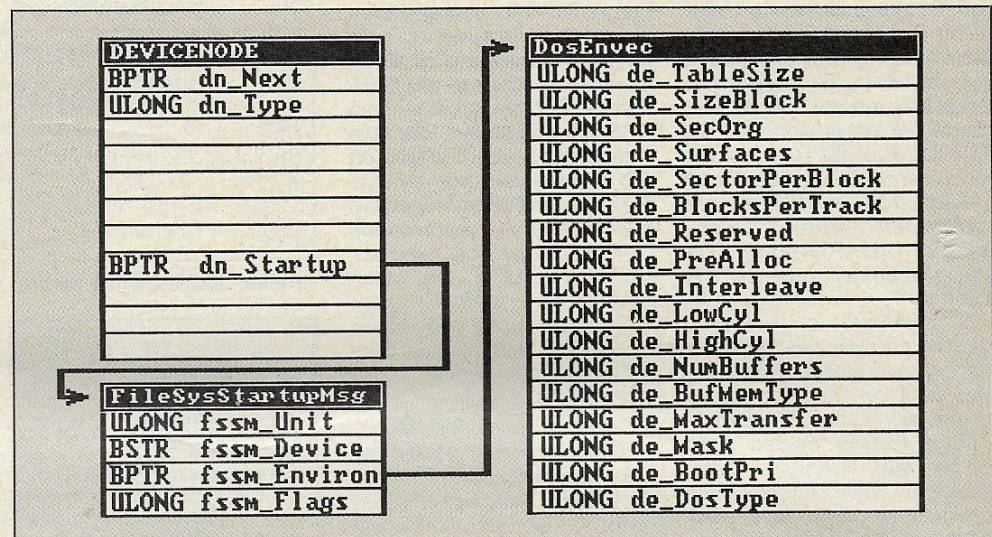


Bild 3. Jede Device-Node enthält einen Zeiger auf eine »FileSystem StartupMessage«

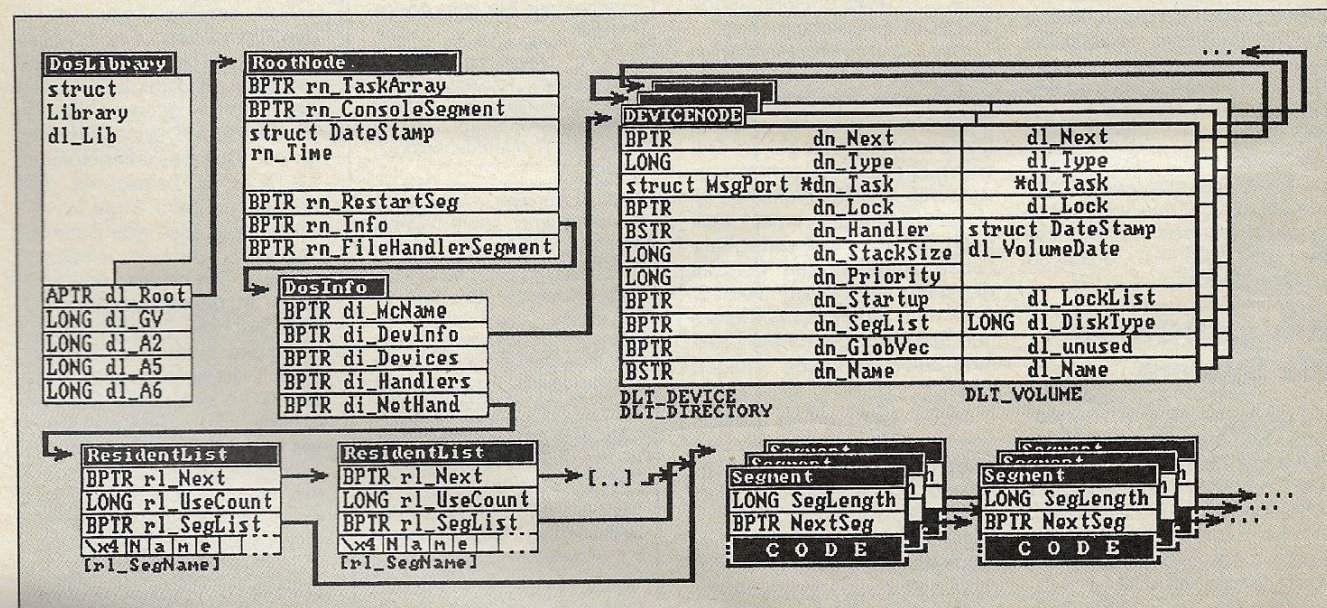


Bild 2. Die verschiedenen Strukturen, die DOS für seine Funktion verwalten muß, sind miteinander verkettet



DLT_DEVICE auf sehr komfortable Weise die Diskettengeometrie eines dateiorientierten Exec-Devices in Erfahrung zu bringen (Bild 3). Der Eintrag »dn_Startup« zeigt in diesem Falle nämlich auf eine Struktur des Typs »FileSysStartupMsg«, mit der die Verbindung zum Exec-Device-System hergestellt wird. In dieser Struktur finden Sie alle Parameter, die zum Aufruf der Exec-Funktion »OpenDevice()« benötigt werden. Außerdem sind alle Laufwerksparemeter übersichtlich in einer Tabelle verzeichnet, auf die mit dem Eintrag »fssm_Envirn« verwiesen wird. Diese Tabelle hat den Namen »DOS Envec« und enthält nach der Größenangabe »de_TableSize« die Parameter, die dem Device physikalisch oder durch die »MountList« vorgegeben sind.

Bis jetzt haben wir nur die DOS-Objekte behandelt, die Amiga-DOS ständig im Speicher hält. Es wäre nicht sehr sinnvoll, wenn die beiden noch verbleibenden DOS-Strukturen, die Directories und Files, ebenfalls ständig im Speicher gehalten werden sollten. Amiga-DOS »weiß« also zunächst nicht, welche Objekte auf einem Volume eingetragen sind. Den »Schlüssel« zu diesen Objekten erhalten wir erst, wenn ein Zugriff auf sie vorgenommen werden soll. Dieser Schlüssel ist eine weitere DOS-Struktur, die den treffenden Namen »FileLock« hat. Ein Lock regelt die Zugriffsberechtigung auf ein Objekt. Ein Lock erhalten Sie übrigens nicht nur für Files, wie man aufgrund des Namens vermuten könnte. Sie können ebenso einen Lock für ein Directory erhalten. Auch hier müssen wir uns wieder im Rahmen unseres Multitasking-Konzeptes die Frage stellen, wer auf diese Objekte in welcher Weise zugreifen darf. Es wäre sicher nicht sehr sinnvoll, wenn ein Task gerade in einem File lesen wollte, während ein anderer diese Datei beschreibt oder gar löscht. Solche »Mißverständnisse« müssen unbedingt vermieden werden.

Schauen wir uns dazu die Struktur FileLock an (Bild 4).

Über den Eintrag »fl_Link« sind die Locks — genau wie die DeviceNodes — in einer Liste verkettet. »fl_Key« bezeichnet den Diskettenblock, auf dem das Objekt gespeichert ist, dazu später noch mehr.

Interessant für unser Problem ist der Eintrag »fl_Access«. Hier sind zwei Typen des Zugriffs erlaubt. Der sogenannte »SHARED_LOCK« (-2) erlaubt es dem Inhabertask, auf das Objekt zuzugreifen ohne andere Tasks auszuschließen. Erst mit einem »EXCLUSIVE_LOCK« (-1) erlangt ein Task die alleinige Zugriffsberechtigung auf das Objekt. Ein EXCLUSIVE_LOCK wird aus diesem Grunde auch nur dann erteilt, wenn bislang kein anderer Task einen Lock auf das Objekt innehat.

Somit kann es zu keinen Mißverständnissen kommen. Die Bedeutung der letzten beiden Einträge ist ebenfalls einleuchtend: »fl_Task« zeigt auf den MessagePort des Handler-Prozesses, also des »File System«-Prozesses des zugehörigen Devices. »fl_Volume« schließlich ist ein BPTR auf die DeviceNode des Volumes, das dieses Objekt enthält.

In Abhängigkeit der offenstehenden Locks »merkt« sich Amiga-DOS ein bestimmtes Volume. Solange noch mindestens ein Lock offensteht, entfernt es auch nicht die DeviceNode-Struktur des Volumes, wie es sonst bei einem Diskettenwechsel geschehen

Diskettenwechsel

würde. Sobald das Volume entnommen wird, auf dem sich unser Objekt befindet, werden alle noch existierenden Locks im Eintrag »dl_LockList« der DeviceNode verkettet. Wenn ein Anwenderprogramm nun wieder über Amiga-DOS auf das Objekt zugreifen möchte, muß der Benutzer zunächst das Vo-

lume einlegen, worauf ihn Amiga-DOS mit dem wohlbekannten Requester »Please insert volume...« hinweist.

Mit Hilfe der FileLock-Struktur haben wir also eine erste Möglichkeit kennengelernt, den Zugriff auf Directory und Files zu regeln. Besonders viele Informationen über die uns interessierenden Objekt konnten wir jedoch bislang noch nicht in Erfahrung bringen. Ein FileLock ist ja auch erst der Schlüssel zu diesen Objekten. Er öffnet uns die Türen zu weiteren Informationen (Bild 5). Die nächste Struktur in diesem Zusammenhang ist der »FileInfoBlock«, mit dem sich alle Eigenschaften unserer Objekte auslesen lassen. Ein FileInfoBlock baut auf der FileLock-Struktur auf. Genauer gesagt, wir müssen zunächst Inhaber eines Locks sein, bevor wir in Besitz eines FileInfoBlocks kommen können. Haben wir einen solchen dann erfolgreich erhalten, können wir aus ihm alle gewünschten Informationen über Directories oder Files erhalten.

Die Inhalte des FileInfoBlocks erklären sich größtenteils von selbst; zu bemerken wäre, daß der Name eines Files oder Directories momentan maximal 30 Zeichen umfassen darf, der Kommentar zu einem File maximal 80 Zeichen. Die Unterstruktur »DateStamp« enthält einen vom Amiga-DOS oft verwendeten »Datumsstempel«, hier drückt er den Zeitpunkt der letzten Änderung eines Files oder das Erstellungsdatum eines Directories aus. Diese Zeitangaben sind immer relativ zum 1. Januar 1978, 00:00:00 Uhr zu interpretieren.

Der Typ des Objekts ist aus »fib_DirEntryType« ersichtlich, positive Werte bezeichnen ein Directory, negative dagegen ein File. Eine analoge Informationsstruktur für Volumes finden wir in der Struktur »InfoData«. Der Schlüssel hierzu ist ein Lock auf irgendein File oder

Directory des zu untersuchenden Volumes.

Wir können nun DOS-Objekte für bestimmte Zugriffsarten reservieren und Informationen über sie auslesen. Die Hauptarbeit des Amiga-DOS besteht jedoch im Lesen und Schreiben von Daten in Files. Dazu müssen wir die Files zunächst »öffnen«. Wir erhalten dann als Ergebnis eine »FileHandle«-Struktur, die uns weitere Operationen ermöglicht. Ein FileHandle wird entweder zum Lesen und Schreiben eines existierenden Files (MODE_OLDFILE 1005) oder eines neu anzulegenden (MODE_NEWFILE 1006) erteilt. Ein weiterer Zugriffsmodus, MODE_READWRITE (1004), öffnet ein bereits existierendes File zum Lesen und Schreiben, wobei von anderen Tasks keine Schreibzugriffe mehr vorgenommen werden dürfen. Operationen zum Öffnen, Lesen, Schreiben und Schließen von Files stellt uns die DOS-Library zur Verfügung. Der FileHandle wird dabei intern vom Amiga-DOS zur Prozeßkommunikation benutzt.

Damit haben wir herausgearbeitet, welche Objekte wir zum Aufbau eines hierarchischen Filesystems benötigen. Was uns noch fehlt, sind die Dienstprogramme, die das Handling dieser Objekte übernehmen. Wenn Sie zu unserem eingangs besprochenen Schaubild (Bild 1) zurückblättern, wird Ihnen auffallen, daß uns noch das Bindeglied zwischen dem DOS-System und den Exec-Routinen fehlt. Diese sogenannten DOS-Handler-Prozesse sollen das Thema unseres nächsten Abschnitts sein.

Prozeßkonzept

In den vorherigen Kursteilen haben wir Ihnen das Exec-Task-Modell und sein Kommunikationssystem vorgestellt. Jedes Programm, das auf dem Amiga zur Ausführung gebracht werden soll, wird unter Zuhilfenah-

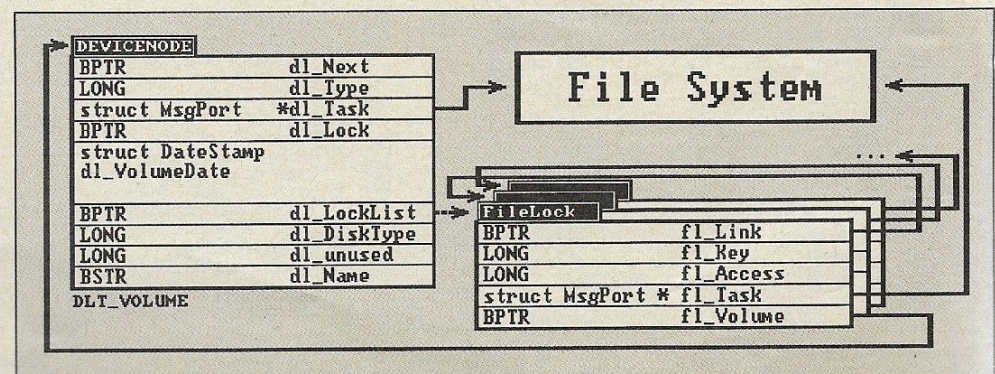


Bild 4. Das Filesystem im Zusammenhang mit Locks

me dieses Konzepts in die Multitasking-Umgebung eingebunden. Wir hatten damals auch die Einschränkung gemacht, daß Tasks nicht in der Lage sind, auf DOS-Funktionen zurückzugreifen. Dies soll aber nicht bedeuten, daß wir zur Anwendung des DOS-Systems ein völlig neues Task-Modell entwerfen müßten.

Amiga-DOS überläßt die Verwaltung seiner Prozesse voll und ganz dem Exec. Auch

Nachrichtensystem abgewickelt. Wie es genau abläuft, wollen wir in einem späteren Abschnitt besprechen. Gleich darauf findet sich ein Hinweis auf Programmcode und Daten des Prozesses. Ein Amiga-Programm ist stets so beschaffen, daß es weder an einer bestimmten Stelle im Arbeitsspeicher noch in einem zusammenhängenden Speicherbereich geladen sein muß. So kann es durchaus sein, daß der Pro-

men wollen wir hier noch besprechen.

»pr_WindowPtr« ist zuständig für die Requester-Ausgabe, die bei Fehlersituationen vom Amiga-DOS ausgelöst wird. Normalerweise ist dieser Wert Null und alle Requester erscheinen auf dem Workbench-Screen. Sie können diesen Eintrag mit -1 überschreiben, dann werden alle Requester-Meldungen für diesen Prozeß ausgesetzt und das Programm er-

art des CLI. Ein CLI-Prozeß kann nämlich auch im Hintergrund ablaufen, der Eintrag »cli_Background« ist in diesem Fall ungleich Null. Ein CLI ist normalerweise »interaktiv«, das heißt, es macht Ausgaben und akzeptiert Eingaben durch den Benutzer. In diesem Falle ist »cli_Interactive« ungleich Null und »cli_Prompt« ist ein BSTR, der den Benutzer zur Eingabe von Befehlen auffordert. Mit dem Kommando »PROMPT« wird dieser manipuliert.

Die Rückgabewerte der KommandoS werden mit drei Variablen verwaltet. »cli_ReturnCode« hält den ersten Fehlercode. So gibt ein erfolgreich beendetes Kommando einen Fehler von Null (RETURN_OK) zurück. Ein fehlerhaft beendeter Aufruf dagegen gibt Werte größer Null zurück. Je nach Schwere des Fehlers ist dies eine 5 (RETURN_WARN), eine 10 (RETURN_ERROR) oder eine 20 (RETURN_FAIL). Mit dem Eintrag »cli_FailLevel« wird festgelegt, ab welchem Fehlercode aufwärts die Kommandofolge abgebrochen werden soll, normalerweise wird er durch das CLI-Kommando »FAILAT« gesetzt. Genauere Fehlerdefinitionen beinhaltet »cli_Result2«, die mit der Library-Funktion »IoErrr« ausgelesen werden. Der Eintrag »cli_SetName« ist ein BSTR auf den Namen des aktuellen Directory. Der BPTR auf die dazugehörige Lock-Struktur findet sich übrigens in der Prozeß-Struktur im Eintrag »pr_CurrentDir«. Beide werden mit dem CLI-Befehl »CD« manipuliert.

»cli_CommandDir« ist auch ein BPTR, und zwar auf eine »PathList«-Struktur, die den Suchpfad für die Kommando-Directories kennzeichnet. Sie kann über den CLI-Befehl »PATH« verändert werden. Wird ein aufgerufener Befehl nicht im aktuellen Verzeichnis gefunden, so geht Amiga-DOS diese Liste durch, bis der Befehl gefunden wurde. Die Liste ist über »pl_NextPath« verkettet, »pl_PathLock« zeigt auf eine FileLock-Struktur des betreffenden Directory. Bleibt nur noch der Eintrag »cli_DefaultStack« in der CommandLine-Interface-Struktur, der die Größe des mit dem CLI-Kommando »STACK« bestimmten Stackbereiches in Langworten ausdrückt. Damit wäre die Struktur der DOS-Prozesse besprochen. In der nächsten Ausgabe des AMIGA-Insider-Kurses fahren wir mit der Beschreibung des Amiga-DOS fort. ■

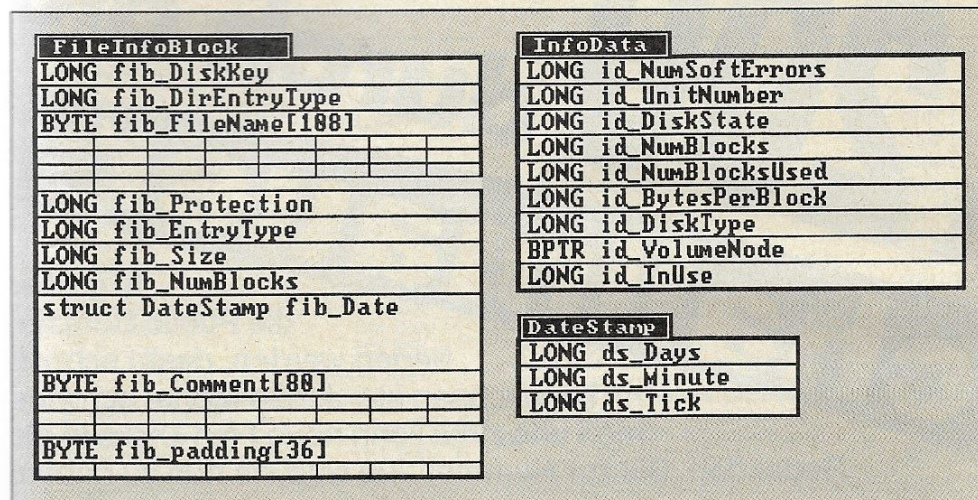


Bild 5. Zu jeder Datei werden Informationen in diesen drei Strukturen aufbewahrt

stimmt es nicht ganz, daß reine Exec-Tasks keine DOS-Funktionen aufrufen dürfen. Wie sollten schließlich sonst die Prozesse erzeugt werden? Der Zugriff auf Funktionen zum Laden von Programmcode und zur Erzeugung von Prozessen ist also auch den Tasks erlaubt. Der feine Unterschied zwischen Task und Prozeß liegt in einer Erweiterung der Task-Struktur, die den nackten Task mit einigen Zusatzinformationen ausstattet. Schauen Sie sich dazu mal die »Prozeß«-Struktur an `dosextens.h`. Da haben wir zunächst die übliche Task-Struktur »pr_Task«, wie zu erwarten war.

Diesen Teil verwaltet Exec, damit werden alle DOS-Prozesse Bestandteil des Multitaskings. Der Rest wird ausschließlich vom DOS verwaltet. »pr_MsgPort« ist eine MessagePort-Struktur, die Ihnen ebenfalls aus den vorherigen Kursteilen bekannt sein sollte. Hier findet die Kommunikation des Prozesses mit dem Betriebssystem und den Anwenderprogrammen statt. Dabei bedient sich der Prozeß einmal des konventionellen Exec-Message-Systems, zum anderen wird hier auch noch ein spezielles, ausschließlich von Amiga-DOS verwendetes

grammcode unseres Prozesses beim Laden in mehrere Stücke zerlegt wird. Diese als »scattered loading« bezeichnete Lademethode sorgt für eine äußerst effiziente Speicherausnutzung. Um die Teile dennoch wie ein zusammenhängendes Ganzes zu verwalten, sind die einzelnen Stücke in einer Liste, der »Segment-Liste« verbunden. Der BPTR »pr_SegList« der Prozeß-Struktur weist auf ein Zeigerfeld mit dem Namen »SegArray«. In diesem Feld befindet sich auf vierter Position (SegArray[3]) ein BPTR auf das erste Element der Segment-Liste. Diese ist, wie könnte es

Scattered loading

auch anders sein, über BPTR verkettet. »NextSeg« zeigt also stets auf das nächste Segment, ist dieser Zeiger gleich Null, so war es das letzte Segment. Die Größe des Segments ist (in Byte) im Offset -4 zum Segment-Pointer abgelegt. Ab Offset 4 beginnen die eigentlichen Daten. Die weiteren Einträge der Prozeß-Struktur werden vom Amiga-DOS intern benötigt. Alle in ihnen enthaltenen Informationen können über Library-Funktionen ausgelesen werden. Lediglich zwei Ausnah-

hält ohne Rückfrage einen Fehlercode. Weitaus interessanter ist es aber, hier den Pointer auf ein eigenes Window einzusetzen, wie er von der Intuition-Funktion »OpenWindow()« zurückgegeben wird. Jetzt erscheinen nämlich alle Requester-Meldungen auf dem Screen dieses Windows. Für eine spezielle Art von Prozeß, nämlich die CLI-Prozesse, verweist der BPTR »pr_CLI« auf eine Struktur namens »CommandLineInterface«. Hier befinden sich weitere spezifische Informationen. Wir finden insgesamt vier FileHandles für die Ein- und Ausgabe. Dies sind »cli_StandardInput« und »cli_StandardOutput«, die sich auf das CLI-Fenster beziehen, »cli_CurrentInput« und »cli_CurrentOutput« dagegen beziehen sich auf die I/O-Kanäle, die beim Kommandoaufruf mit »<« und »>« umgeleitet werden können. »cli_CommandName« ist ein BSTR auf die aktuelle Kommandozeile, »cli_Module« zeigt dabei auf die Segment-Liste des geladenen Kommando S. Mit »cli_CommandFile« wird eine Kommando-Datei (Batch-File) benannt, die gerade über das CLI-Kommando EXECUTE ausgeführt wird. Drei weitere Einträge bestimmen die Betriebs-