robotron

Anwendungsbeschreibung für den Bürocomputer AS 120.16

Anwendungsbeschreibung MUTOS 8000

Streichhahn

robotron

Anwendungsbeschreibung

fuer den Buerocomputer A5120.16

Anwendungsbeschreibung MUTOS 8000

VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt
Stand: 12/85

Die vorliegende Dokumentation entspricht dem Stand 12/85.

Im Interesse einer staendigen Weiterentwicklung werden alle Leser gebeten, ihre Vorschlaege bzw. Hinweise dem

VEB Robotron-Buchungsmaschinenwerk
9010 Karl-Marx-Stadt
Annabergerstr. 93

mitzuteilen.

Fuer das Betriebssystem MUTOS 8000 wurden folgende Dokumentationen erarbeitet:

- Anwendungsbeschreibung MUTOS 8000 Teil I,
  - Abschnitt 1 Kommandos
  - Abschnitt 2, 3 Systemrufe, Bibliotheksfunktionen
  - Abschnitt 4, 5, 7, 8 Special Files, Fileformate, Makropakete, Systemunterstuetzung

- Systemhandbuch MUTOS 8000 Teil II,
  - Abschnitt 1 Kommandointerpreter Shell
  - Abschnitt 2 Editor
  - Abschnitt 3 Testhilfsprogramm adb

- Systemhandbuch MUTOS 8000 Teil III,
  - Abschnitt 1 Textverarbeitung nroff

- Sprachbeschreibungen
  - Assemblersprachbeschreibungen U8000
  - Sprachbeschreibung C-Sprache

Das Sachwortverzeichnis ist als Anlage in der Anwendungsbeschreibung enthalten.

Zum Programmentwickeln und -Testen fuer den Prozessor U8000 werden zusaetzlich folgende Dokumentationen angeboten:

- Monitorsystem MON8000
- CROSS-Software U8000 zum Betriebssystem UDOS
- Assembler U8000ASM
- Binder ZLINK
- Absolutbinder IMAGER
- Preprozessor INCLUDE

Inhaltsverzeichnis Seite

1. Systemuebersicht

1.1. Einfuehrung

MUTOS 8000 ist ein interaktives Timesharing-Betriebssystem fuer den Buerocomputer A5120.16. Der Buerocomputer A5120 wurde durch ein Leiterkartenpaar (EM 256) erweitert. Der Erweiterungsmodul besteht aus einem 16 - Bit - Mikroprozessor U 8001 und einem 256 KByte RAM-Speicher. Der Buerocomputer A5120.16 arbeitet unter MUTOS 8000 im 16-Bit-Modus. Die bekannten Betriebssysteme SIOS 1526, SCP 1526 und UDOS 1526 koennen auf dem A 5120.16 auch eingesetzt werden. Der Rechner arbeitet dann im 8-Bit-Modus.
MUTOS 8000 bietet den Nutzern eine Programmierumgebung, die modernsten Anspruechen genuegt. Dies gilt sowohl fuer die Realisierung des Mensch-Maschine-Dialogs durch einen leistungsfaehigen Kommandointerpreter, universell nutzbare Kommunikationsmechanismen und eine sehr dialogfreundliche Gestaltung aller Systemprogramme, die eine baukastenartige Verknuepfung der einzelnen Leistungen zulassen, als auch fuer die vorhandenen Programmierwerkzeuge. Die Arbeit mit Files (Dateien) ist problemlos. Der Weg von der Programmuebersetzung bis zum Programmstart gestaltet sich aeusserst einfach. Als hoehere Programmiersprache ist zunaechst nur die Sprache C implementiert. Durch deren Ausdrucksmittel und Effizienz wird ein Assemblereinsatz auf Nutzerprogrammebene ueberfluessig. Dem Programmierer steht auch eine umfangreiche Systembibliothek zur Verfuegung.
Durch diese Programmierumgebung wird der Anwender weitgehend von Nebensaechlichem entlastet, so dass mehr als bei allen bisher existierenden Betriebssystemen das Hauptaugenmerk den zu loesenden Anwenderproblemen gelten kann.
Ein besonderer Vorzug von MUTOS 8000 ist auch, dass der Nutzer dieselbe Programmierumgebung auch im System PSU (auf ESER-Anlagen), INMOS (auf SKR-Anlagen) und bei MUTOS - 1630 (auf K1630) vorfindet.
Fuer einen Einsatz in der Prozessdatenverarbeitung ist MUTOS 8000 nicht geeignet, da keine Echtzeitunterstuetzung existiert.

1.2. Bestandteile des Betriebssystem MUTOS 8000

1.2.1. Softwarekomponenten

Das Betriebssystem MUTOS 8000 umfasst folgende Programmgruppen:

- MUTOS 8000-Kern mit folgenden Aufgaben:
  - Nutzerprogrammorganisation
  - E/A - Organisation
  - Filesystem

- Dienstprogrammgruppen fuer folgende Aufgabenkomplexe:
  . Filemanipulation
  . Manipulation von Directories (Dateiverzeichnissen) und Filenamen
  . Abarbeitung von Nutzerprogrammen
  . Nutzerkommunikation
  . Programmentwicklung
  . automatische Textverarbeitung
  . Bearbeitung grosser Datenmengen
  . Steuerung des Nutzerzugriffs und Einstellen von Terminalfunktionen
  . Statusabfragen
  . Wartung und Betrieb des Systems
  . Leistungsabrechnung

- C - Compiler

- Bibliotheken fuer
  . E/A - Arbeit
  . Programmorganisation
  . mathematische Funktionen

1.2.2. Dokumentationen

Dem Anwender stehen folgende Dokumentationen zur Verfuegung:

- Anwendungsbeschreibung

- Systemhandbuch Teil I
  - Abschnitt 1 Kommandos
  - " 2 Systemrufe
  - " 3 Bibliotheksfunktionen
  - " 4 Special Files
  - " 5 Fileformate
  - " 7 Makropakete, Sprachkonventionen
  - " 8 Systemunterstuetzung

- Systemhandbuch Teil II
  - Abschnitt 1 Kommandointerpreter Shell
  - " 2 Editor
  - " 3 Testhilfsprogramm

- Systemhanbuch Teil III
  - Abschnitt 1 Textverarbeitung nroff

- Sprachbeschreibungen
  - Assemblersprachbeschreibung U 8000
  - Sprachbeschreibung C-Sprache

1.3. Systemphilosophie

MUTOS 8000 ist als interaktives Teilnehmersystem konzipiert. Ein wichtiges Ziel von MUTOS 8000 ist es, dem Nutzer fuer die Bearbeitung seiner Aufgaben eine nutzerfreundliche Programmierumgebung zu schaffen.
Im Gegensatz zu vielen anderen Betriebssystemen, die eine sehr komplexe und oft unuebersichtliche Prozess- und Speicherverwaltung oder eine Vielzahl von Filezugriffsroutinen unterstuetzen, wird bei MUTOS 8000 in diesen Bereichen Einfachheit und gute Verstaendlichkeit angestrebt. Dies wird durch die praktisch ausschliessliche Verwendung einer hoeheren Programmiersprache, leicht zu benutzende Systemrufe und auf diesen aufbauende Systembibliotheksroutinen erreicht.
Besondere Beachtung verdient in einem Dialogsystem der Kommandointerpreter. Dort sind gute Verstaendlichkeit auch fuer den Anfaenger, gepaart mit hoher Leistungsfaehigkeit fuer anspruchsvolle Nutzer realisiert. Desweiteren beruecksichtigen die Systemprogramme die Erfordernisse des Dialogs. Zur Realisierung dieser Leistungen wurde bei MUTOS 8000 folgendermassen verfahren:

- Implementierung eines hierarchischen Filesystems, welches um eine beliebige Anzahl von Datentraegern erweitert werden kann

- Gleichartige Behandlung von File, Geraete- und Interprozess-E/A (Die Interprozess-E/A dient der Kommunikation zwischen Prozessen und wurde auf einfache Weise ueber einen besonderen Mechanismus im Filesystem, das Pipelining, realisiert.)

- automatische Speicherzuordnung und Prozessauslagerung

- Implementierung einer effektiven Kommandosprache mit den Ausdrucksmitteln einer hoeheren Programmiersprache.

- Es existiert eine grosse Anzahl von Dienstprogrammen, die durch universell nutzbare Kommunikationsmechanismen des Systems und ihre angepassten Dialogeigenschaften leicht untereinander kombiniert und in die Problemloesung des Anwenders eingebunden werden koennen.

- Anwendung der hoeheren Programmiersprache C in der Systemprogrammierung, wo bei bisherigen Betriebssystemen Assemblersprachen angewendet wurden. (Nur etwa 10% des MUTOS 8000-Kerns sind in Assembler, der Rest ist in C geschrieben.)

- Erreichen einer guten Portabilitaet des gesamten Systems (fast alle Systemprogramme sind in C geschrieben).

1.4. Systempflege

Das Betriebssystem MUTOS 8000 mit seinen Komponenten wird staendig gepflegt und erweitert. Dabei wird auf die effektive Nutzung der Hardwarekomponenten sowie Erweiterung der Systemleistungen und Anwendungspakete orientiert. Zur Kennzeichnung der aktuellen MUTOS-Version wird eine Versionsnummer (fuer Boot- und Root-Diskette) eingefuehrt. Der Nutzer wird ueber jede Aenderung im Rahmen der SIOS-Informationsschrift informiert.

2. Anwendungsbedingungen

MUTOS 8000 benoetigt folgende Mindestkonfiguration:

Buerocomputer A5120 mit

- 64 KRAM
- Bildschirm 80 x 24 Zeichen
- Tastatur K7637 (K7636 moeglich)
- Folienspeicher 3- 4 Laufwerke (K 5602 oder/und K 5600.20)
- Seriendrucker (1157/1152; SIO/PIO-Interface)
- Erweiterungsmodul EM256

3. Systemanlauf

Fuer den Systemanlauf von MUTOS 8000 wird eine Ladediskette (Boot) benutzt.
Nach dem Laden des Boot-Systems koennen auf dieser Ebene verschiedene Systemdienste wie

- Formatieren und Kopieren von Boot-Disketten und
- Formatieren und Kopieren von MUTOS-Disketten

ausgefuehrt werden.
Das Betriebssystem MUTOS 8000 wird von hier aus mit dem Laden der Root-Diskette erreicht. Dazu wird das Kommando

BOOT MUTOS

benutzt, wobei zwischen Druckervarianten SIO (Standardvariante) und PIO (Kommando BOOT MUTOS.PIO) gewaehlt werden kann. Nach dem Systemanlauf MUTOS 8000 hat sich der Nutzer mit seinem Namen anzumelden. Er kann als Superuser mit allen Zugriffsrechten oder als "normaler" Nutzer mit eingeschraenkten Zugriffsrechten arbeiten (Login-Name root/user).
Nach diesem Login-Prozess wird das Arbeitsdirectory (HOME) des Nutzers erreicht. Das Systemdatum ist zu aktualisieren. Im allgemeinen wird dann das Filesystem, dass jetzt nur aus der Root-Diskette besteht, mit dem Kommando **mount** erweitert und so weitere Disketten eingegliedert.
Lesen Sie bitte dazu die Beschreibungen boot(8), login(1), mount(1), sowie form(8), copy(8) und filesys(5) im Systemhandbuch Teil I.

4. Leistungen des MUTOS 8000 - Kerns

Der MUTOS 8000-Kern koordiniert das Abarbeiten aller zu bearbeitenden Prozesse durch eine zentrale Prozess - und Speicherverwaltung, deren Aufgabe es ist, die Systemleistungen effektiv zu lenken. Ausserdem wird durch den MUTOS 8000-Kern die Arbeit der Nutzer mit dem Filesystem realisiert. Dazu kommt als dritter grosser Aufgabenkomplex des MUTOS 8000-Kerns die Organisation der Ein- und Ausgabe und die Behandlung aller Interrupts. Es existieren Geraetetreiber fuer alle unter Pkt.2 (Geraetekonfiguration) aufgefuehrten externen Geraete sowie ein blockorientiertes E/A-System, das die Verbindung zwischen Geraetetreiber und Filesystem bildet.
Auf die Arbeit mit Prozessen und dem Filesystem in MUTOS 8000 wird im folgenden naeher eingegangen. Zum blockorientierten E/A-System werden in Verbindung mit dem Filesystem einige Aussagen gemacht.

4.1. Das MUTOS 8000-Filesystem

MUTOS 8000 bietet dem Nutzer zur Arbeit im Dialog ein transparentes, leicht zu verwaltendes und dabei sehr leistungsfaehiges hierarchisch strukturiertes Filesystem.
Durch einfach zu benutzende Systemrufe und einen Satz darauf aufbauender Bibliotheksroutinen wird die Arbeit mit Files in Programmen erleichtert. Zur Unterstuetzung der Arbeit im Dialog stehen zahlreiche Dienstprogramme zur Verfuegung, durch welche man Files manipulieren oder Informationen ueber das gesamte Filesystem oder Teile davon erhalten kann. Es existiert die Moeglichkeit, das Wurzelfilesystem (Systemresidenz) durch Eingliedern weiterer Filesysteme zu erweitern (mount-Systemruf), wobei die Filesysteme der einzelnen Datentraeger getrennt verwaltet werden. Das gesamte Filesystem wird sich also in der Regel aus mehreren Filesystemen zusammensetzen und so die gesamte Filehierarchie festlegen. Durch die getrennte Verwaltung wird ein Ausgliedern einzelner Filesystemen erleichtert (umount - Systemruf). Die Implementation erlaubt ein praktisch wartungsfreies Arbeiten mit dem Filesystem (kein Verdichten notwendig).
Alle Geraete wurden im Filesystem integriert, die Arbeit mit ihnen unterscheidet sich nicht von der Arbeit mit regulaeren Files. Ein gut handhabbarer Schutzmechanismus ermoeglicht es, Files vor unerlaubtem Zugriff zu schuetzen. In MUTOS 8000 gibt es drei Arten von Files, die nun erlaeutert werden sollen.

4.1.1. Regulaere Files

Regulaere Files enthalten Informationen, die vom Nutzer darin abgelegt wurden (zum Beispiel Quell- oder Objektprogramme). Vom Betriebssystem wird keine bestimmte Filestruktur verlangt. Ein Textfile besteht einfach aus einer Zeichenkette. Zeilen sind dabei durch das uebliche Zeilenendekennzeichen (hex. 0A) getrennt. Binaere Programme sind Folgen von Worten, wie sie in den Speicher geladen werden muessen, wenn das Programm ausgefuehrt werden soll. Viele Dienstprogramme arbeiten mit staerker strukturierten Files. So erzeugt zum Beispiel der

Assembler Objektfiles in einem bestimmten Format, das vom Lader verlangt wird. Die Struktur von Files wird aber immer von den Programmen, die mit ihnen arbeiten, nicht aber vom System bestimmt.

4.1.2. Directories

Directories sind Verzeichnisse von Filenamen mit zugehoerigen i-Nummmer. Ueber letztere sind die zu einem File gehoerenden Informationen, insbesondere die Adressen der einzelnen Bloecke eines Files, zu erreichen. Sie stellen somit die Verbindung zwischen den Filenamen und den Files selbst her. Durch sie wird folglich dem Filesystem eine konkrete Struktur aufgepraegt.
Jeder Nutzer kann eigene Directories und auch Subdirectories anlegen, die Gruppen von Files unter geeigneten Gesichtspunkten zusammenfassen.
Directories werden vom System wie regulaere Files behandelt, mit der Ausnahme, dass sie nur durch priviligierte Programme beschrieben werden koennen, so dass also das System ihren Inhalt kontrolliert.
Von MUTOS 8000 wird eine Menge von Directories fuer den eigenen Gebrauch verwaltet. So zum Beispiel das Root-Directory, das der Ursprung (die Wurzel) des baumstrukturierten Filesystems ist. Durch schrittweises Bewegen durch die Filehierarchie ist vom Root-Directory aus jedes File erreichbar. Andere Systemdirectories enthalten zum Beispiel alle Dienstprogramme oder Files, die der Systemverwaltung dienen. Wie spaeter zu sehen sein wird, sind diese Festlegungen modifizierbar, sollten aber nach einmaliger Festlegung moeglichst beibehalten werden.
Filenamen bestehen aus maximal 14 beliebigen ISO-Zeichen. Will das System einen Zugriff zu einem bestimmten File realisieren, so muss der Pfadname des Files angegeben werden. Dieser Pfadname ist eine Folge von Directorynamen, die durch "/" getrennt sind, und mit dem Filenamen enden. Beginnt die Folge mit "/", wird die Suche beim Root-Directory begonnen. So sucht zum Beispiel das System beim Pfadnamen /alpha/beta/gamma zunaechst im Root-Directory das Directory alpha, dort das Directory beta und schliesslich in diesem gamma. Dabei kann das File gamma ein regulaeres File, ein Directory oder ein Special-File sein.

Mit "/" kann das Root-Directory selbst angesprochen werden. Faengt der angegebene Pfadname nicht mit "/" an, beginnt das System die Suche im aktuellen Directory (Arbeitsdirectory) des Nutzers. Zum Beispiel bezeichnet alpha/beta dann den Filenamen beta im Subdirectory alpha des aktuellen Directories. Das aktuelle Directory kann vom Nutzer selbst festgelegt werden, sofern dies nicht den Festlegungen fuer die Zugriffsrechte widerspricht (siehe Schutzmechanismus, 4.1.5.).

Ein Nicht-Directory-File kann in verschiedenen Directories des Filesystems einer Diskette unter (moeglicherweise) verschiedenen Namen registriert sein, obwohl es physisch nur einmal existiert. Diese Moeglichkeit wird Linking genannt, entsprechend wird ein Directory-Eintrag fuer Files auch als Link bezeichnet. Alle Links zu einem File haben den gleichen Status. Dieser Eintrag im Directory enthaelt nur den Filenamen und einen Verweis zu den Informationen, die das File beschreiben (Eigentuemer, Erstellungs- und letztes Modifikationsdatum sowie den Verweis zu den einzelnen Bloecken des Files auf der Diskette). So existiert also ein File eigentlich unabhaengig von einem Eintrag in ein Directory, allerdings wird sein Speicherbereich freigegeben, wenn fuer das File kein Eintrag mehr existiert.

Jedes Directory hat von Beginn an zwei Eintragungen: die Namen "." und "..". Der Name "." verweist auf das Directory selbst. Ein Programm kann das aktuelle Directory unter der Bezeichnung "." lesen, ohne den vollstaendigen Pfadnamen zu kennen. Der Name ".." verweist zu dem Directory, in welchem es selbst definiert ist. Die Directory-Struktur ist eine Baumstruktur. Abgesehen von den speziellen Eintragungen "." und ".." gilt, dass fuer jedes Directory genau ein Eintrag in einem anderen Directory existieren muss. Das Ziel dieser Festlegung ist,das Schreiben von Programmen, die Teilbaeume der Directory-Struktur behandeln, zu vereinfachen und zu verhindern, dass sich Teile der Hierarchie verselbstaendigen.

4.1.3. Special Files

Special Files sind ein Bestandteil des MUTOS 8000-Filesystems, der es verdient, besonders hervorgehoben zu werden. Zu jedem E/A-Geraet, das von MUTOS 8000 unterstuetzt wird, gehoert ein solches Spezial-File. Diese Files koennen wie regulaere Files beschrieben oder gelesen werden. Dies bewirkt dann ein Lesen von dem oder ein Schreiben auf das ihnen zugeordnete Geraet.

Fuer jedes existierende Spezial-File existiert im Directory /dev ein Eintrag. Wie bei regulaeren Files koennen aber auch Links in anderen Directories zu diesem File existieren. Es gibt Spezial-Files fuer jedes Disketten-Laufwerk, den Drucker, die Bedieneinheit und auch fuer den Hauptspeicher (wobei natuerlich durch genau definierte Zugriffsrechte ein unerlaubter Zugriff, z.B. zum Hauptspeicher, vermieden werden muss). Durch Schreiben auf /dev/lp kann z.B. eine Ausgabe auf den Drucker realisiert werden. (Lesen ist hier natuerlich unsinnig und fuehrt zu einer Fehlermeldung, da ein entsprechendes Zugriffsrecht sinnvollerweise nicht existiert.)

Der Grund fuer diese Art und Weise des Umgangs mit E/A-Geraeten ist, dass so fuer File- und Geraetenamen die gleiche Syntax und Semantik gueltig und einem Programm, welches als Parameter einen Filenamen erwartet an dieser Stelle auch ein Geraetename uebergeben werden kann (siehe z.B. in Shell bei E/A-Umlagerung). Schliesslich wird auch fuer Special Files derselbe Schutzmechanismus wie fuer andere Files benutzt.

4.1.4. Erweiterungsmoeglichkeiten des Filesystems

In MUTOS 8000 gibt es die Moeglichkeit, ein Filesystem auf einer Diskette durch ein weiteres Filesystem auf einer anderen Diskette so zu erweitern, dass nach aussen hin die Erweiterung nicht mehr erkennbar ist. Um ein Filesystem eingliedern zu koennen, existieren der mount- Systemruf und ein darauf aufgebautes mount- Kommando. Der Systemruf hat die Form:

mount(para1,para2)

Dabei ist:

para1: Der Name eines Spezial-Files, dem eine Diskette mit einem eigenen Filesystem zugeordnet ist.
para2: Der Name eines existierenden Directories im zu erzeugenden Filesystem.

Der mount- Systemruf bewirkt, dass der Eintrag fuer das angegebene Directory nicht mehr zum eigentlichen Directory-Inhalt sondern zum Inhalt des Root-Directories des eingegliederten Filesystems verweist. Es wird also die Filehierarchie erweitert, indem ein Teil davon durch ein Filesystem ersetzt wird, das sich auf einem anderen Datentraeger befindet. Nach dem Eingliedern des zusaetzlichen Filesystems gibt es scheinbar keinen Unterschied zwischen Files im hinzugefuegten und urspruenglichen Filesystem mehr, mit der Ausnahme, dass es Links zwischen Files in verschiedenen Filesystemen nicht geben darf.
In MUTOS 8000 koennen neben dem Root-Filesystem (Root-Diskette) maximal 3 weitere Filesysteme auf unterschiedlichen Disketten eingegliedert werden.

4.1.5. Der Fileschutzmechanismus

Obwohl die Prinzipien des Fileschutzes bei MUTOS 8000 bei der Verwendung "personengebundener Filesysteme" (Disketten), also im Single-User-Betrieb, nicht voll zur Geltung kommen, gibt es trotzdem eine Reihe interessanter Gesichtspunkte.

Jeder Nutzer des Systems hat einen Nutzer-und Gruppenidentifikator (ID), den er normalerweise vom Systemverwalter fuer die gesamte Zeit der Arbeit am System zugeordnet erhaelt. Der Gruppen-ID kann im Zusammenhang mit der kollektiven Bearbeitung groesserer Projekte (z.B. Nutzung einer gemeinsamen Textkonserve) an mehrere Nutzer vergeben werden, waehrend jeder Nutzer einen eigenen Nutzer-ID erhaelt. Legt der Nutzer ein File an, so wird dieses intern mit dem Nutzer- und Gruppen-ID versehen. Ebenso werden fuer jedes File unter anderem 10 Schutzbits angelegt, von denen neun dazu dienen, die Zugriffsrechte Lesen, Schreiben und Ausfuehren fuer den Fileeigentuemer, die anderen Mitglieder seiner Nutzergruppe sowie fuer die restlichen Nutzer des Systems festzulegen.
Jedes Programm laeuft mit dem Kennzeichen des Nutzer- und Gruppen-ID des aktuellen Nutzers (aktueller Nutzer- und Gruppen-ID). Bei Filezugriffen wird der aktuelle Nutzer- und Gruppen-ID mit Nutzer- und Gruppen-ID des Fileeigentuemers (i.a. der Fileersteller, s.a. Dienstprogrammbeschreibung chown) verglichen und daraus die Zugriffsrechte abgeleitet (nach Einordnung des aktuellen Nutzers in Fileeigentuemer, Gruppenmitglied oder restlichen Nutzer).
Ist das 10. Schutzbit eines ausfuehrbaren Files (Programm) gesetzt, tauscht das System waehrend der Zeit der Ausfuehrung dieses Files den aktuellen Nutzer-ID gegen den Nutzer-ID des Programmeigentuemers aus und gestattet sonst dem aktiven Nutzer den Zugriff auf Files, die den gleichen Eigentuemer wie das gerade abzuarbeitende Programm haben.
Der Zweck dieser Einrichtung besteht darin, Nutzern den Zugriff zu bestimmten Files nur ueber bestimmte (meist privilegierte) Programme zu gestatten. Zum Beispiel kann ein Programm ein Abrechnungsfile verwalten, welches nur von diesem Programm selbst (und auf keinem anderen Wege) geaendert werden soll.

Der Austausch des Nutzer-ID ist nur zur Ausfuehrungszeit des Programmes, welches dafuer vorgesehen ist, wirksam. Dieser Mechanismus wird u.a. benoetigt, um alle Nutzerprogramme ausfuehren zu koennen, die bestimmte privilegierte Systemrufe benutzen. Zum Beispiel gibt es einen Systemruf, der ein Directory erzeugt, aber nur vom Super-User (Nutzer fuer den es keine Einschraenkungen in den Zugriffsrechten gibt) ausgefuehrt werden kann.
Wie schon beschrieben, haben Directories von Anfang an die Eintraege "." und "..". Das Dienstprogramm (ein entsprechendes ausfuehrbares File) zum Erstellen neuer Directories gehoert dem Super-User und hat ein gesetztes 10. Schutzbit. Will ein beliebiger Nutzer ein neues Directory erstellen, wird zunaechst seine Berechtigung dafuer ueberprueft und wenn diese besteht, das Directory (durch gesetztes 10. Schutzbit im Super-User Modus des Fileeigentuemers) erzeugt und die Eintraege "." und ".." gemacht.

Da jeder bei seinen eigenen Files dieses Nutzer-ID-Bit setzen kann, ist dieser Mechanismus allgemein verfuegbar. Im System gibt es, wie oben erwaehnt, einen Nutzer-ID (den des Super-Users), fuer den es keine Einschraenkungen in den Zugriffsrechten gibt. Somit koennen auch Programme zur Verfuegung gestellt werden, ohne Einschraenkungen durch Schutzmechanismen beruecksichtigen zu muessen (Super-User-ID wird waehrend deren Ausfuehrung gesetzt).

4.1.6. E/A - Systemrufe

Die Systemrufe zur Ein- und Ausgabe sind so angelegt, dass Unterschiede zwischen den verschiedenen Geraeten und Zugriffsarten weitgehend eliminiert werden. Es gibt keinen Unterschied zwischen wahlfreiem Zugriff und sequentieller Ein- und Ausgabe. Ebenso wird keine logische Satzlaenge vom System unterstuetzt. Die Laenge eines regulaeren Files ist festgelegt durch die Anzahl von Bytes, die es enthaelt. Eine Vordefinition von Filelaengen ist nicht notwendig und auch nicht moeglich.

Im folgenden soll mit Hilfe einiger grundlegender Systemrufe versucht werden, den Gebrauch von Ein- und Ausgaben in MUTOS 8000 zu erlaeutern. Dies geschieht in einer fiktiven C-aehnlichen Sprache, bei der die geforderten Parameter eines Systemrufs ohne genaue Spezifikation verwendet werden. Fuer jeden Systemruf kann es eine Fehlerrueckkehr geben, die hier ebenfalls nicht beruecksichtigt wird.
Um ein File lesen oder beschreiben zu koennen, muss es vorher mit folgendem Systemruf geoeffnet worden sein:

```
filedes = open(name,flag)
```

Dabei ist name der Name des zu oeffnenden Files. Hier kann ein beliebiger Pfadname angegeben werden. Der Parameter flag gibt an, ob das File zum Lesen, Schreiben oder Aendern (Lesen und Schreiben gleichzeitig) geoeffnet werden soll. Den Rueckkehrwert fildes bezeichnet man als Filedescriptor. Dieser ist eine ganze Zahl, die zur Identifikation des Files in den folgenden Rufen (read,write oder andere Filemanipulationen) benutzt wird. Um ein neues File anzulegen oder ein altes vollstaendig zu ueberschreiben, steht der Systemruf

```
fildes = create(name,flag)
```

zur Verfuegung. Durch die angegebene Anweisung wird das File name erstellt, wenn es noch nicht existiert oder ansonsten seine Laenge auf 0 gesetzt. Anschliessend wird das File geoeffnet und wie bei open der Filedescriptor zurueckgegeben und unter fildes gespeichert.

Zunaechst soll nur das sequentielle Beschreiben und Lesen von Files betrachtet werden. Das bedeutet, dass beim naechsten E/A-Ruf bei dem Byte fortgesetzt wird, welches dem zuletzt geschriebenen (bzw. gelesenen) Byte des betreffenden Files folgt. Fuer jedes geoeffnete File existiert ein Zeiger, der vom System verwaltet wird und das naechste zu lesende bzw. zu beschreibende Byte des Files angibt. Ist ein File geoeffnet, kann von folgenden Systemrufen Gebrauch gemacht werden:

```
n = read(fildes,buffer,count)
n = write(fildes,buffer,count)
```

Durch diese Systemrufe werden maximal count Bytes zwischen dem durch fildes spezifizierten File und dem durch buffer angegebenen Byte-Array uebertragen. Im Fall write ist n im Normalfall gleich count (Ausnahmen sind z.B. bei E/A- Fehlern moeglich). Bei einem read -Systemruf ist bei fehlerfreier Ausfuehrung n stets kleiner gleich count. Wird beim Lesen des Files das Fileende ueberschritten, bricht das System die Uebertragung am Fileende ab und n enthaelt die Anzahl der uebertragenen Zeichen. Ist n gleich 0, so ist das Fileende bereits vorher erreicht werden.
Beim Schreiben werden nur die Teile eines Files veraendert, die durch den Stand des write -Zeigers und die angegebene Zeichenzahl betroffen sind. Der Rest des File bleibt unveraendert. Wird dabei das Fileende ueberschritten, so wird das File um soviel Byte wie noetig erweitert.
Um einen wahlfreien Zugriff zu ermoeglichen, ist es nun nur noetig den read - oder write - Zeiger so zu veraendern, dass auf das entsprechende Byte des Files als naechstes zugegriffen wird. Dies geschieht durch

```
location = lseek(fildes,offset,base)
```

Der Zeiger, der zum Filedesciptor fildes gehoert, wird auf eine Position gestellt, die durch offset (Verschiebung) und base angegeben wird. Dabei gibt base an, ob als Bezugspunkt der Fileanfang, der aktuelle Zeigerstand oder das Fileende gilt, waehrend offset die Verschiebung bezueglich dieser Basis definiert.
Offset kann auch negativ sein. Fuer einige Geraete (z.B. Terminal) wird der lseek - Ruf ignoriert. Der aktuelle Stand des read - oder write - Zeigers wird in location zurueckgegeben.
Es gibt noch eine Vielzahl von Moeglichkeiten im System, die zur E/A-Arbeit gehoehren und mit dem Filesystem zu tun haben, hier aber nicht detailliert erlaeutert werden sollen. Dies sind zum Beispiel Rufe zum Schliessen eines Files, des Eigentuemers, Erstellen eines Directories, Erzeugen eines Link zu einem File und Loeschen eines Files. Diese und noch weitere Rufe zu dieser Problematik sind im Systemhandbuch Teil I, Abschnitt 2 Systemrufe Systemrufe, genau beschrieben.

4.2. Das Implementieren des Filesystems

Wie unter 3.1.2. bereits erwaehnt, enthaelt ein Directory-

Eintrag nur den Namen des Files und einen Zeiger zum File selbst.
Dieser Zeiger ist eine ganze Zahl und wird auch i-number (fuer Index-Number) des Files genannt. Soll auf ein File zugegriffen werden, wird seine i-number als Index fuer die entsprechende Systemtabelle ilist) benoetigt. Die i-list ist in einem genau definierten Teil der Diskette enthalten, welche das Directory enthaelt. Ein Eintrag in der i-list wird i-node genannt. Im i-node eines Files befindet sich die Beschreibung des betreffenden Files, die folgende Informationen umfasst:

- Nutzer- und Gruppen-ID des Fileeigentuemers
- die Schutzbits
- die physischen Adressen der zum File gehoerenden Bloecke auf der Diskette
- die Filelaenge
- Datum der Erstellung, der letzten Benutzung und letzten Modifikation
- die Anzahl der Links zum File, d.h. die Anzahl, wie oft dieses File in einem Directory auftritt
- eine Markierung die angibt, ob das File ein Directory, ein regulaeres File oder ein Spezial-File ist.

Ein open - bzw. create - Ruf dient u.a. dazu, den angegebenen Pfadnamen in eine i-number umzuwandeln. Zu diesem Zweck werden nacheinander die angegebenen Directories durchsucht. Ist ein File geoeffnet, sind seine Geraetenummer (gibt das Filesystem an), seine i-number und sein read/write -Zeiger in einem Eintrag in einer hauptspeicherresidenten Systemtabelle gespeichert. Die Nummer des Eintrags gibt den vom open - bzw. creat - Ruf zurueckgegebene Filedescriptor an. So ist eine unmittelbare Verbindung zwischen Filedescriptor und den zum geoeffneten File gehoerigen Informationen gegeben.

Beim Anlegen eines neuen Files wird ein i-node fuer dieses File erzeugt. Ausserdem erhaelt das entsprechende Directory einen Eintrag, der aus Filenamen und i-number besteht. Wird ein Link zu einem bereits existierenden File erzeugt, wird die i-number vom Original-Directory-Eintrag verwendet und der Link-Counter im i- node um 1 erhoeht. Beim Loeschen eines File wird der entsprechende Directory-Eintrag beseitigt und der Link-Counter im i-node um 1 verringert. Wird dieser dabei 0 (kein Directory-Eintrag mit dieser i-number existiert mehr), werden die zum File gehoehrenden Disketten bloecke und der i-node freigegeben,
Das Speichervolumen ist auf allen Disketten, die Filesysteme enthalten, in Bloecke zu 512 Byte unterteilt. Diese werden logisch von 0 bis zu einer initialisierungs- und geraeteabhaengigen oberen Grenze adressiert. Im i-node eines jeden Files ist Platz fuer 13 Blockadressen. Fuer Nicht-Spezial-Files verweisen die 10 ersten dieser Blockadressen zu den 10 ersten Bloecken des Files. Umfasst das File mehr als 10 Bloecke, so verweist die naechste Blockadresse zu einem Block, der die Adressen der (bis zu) 128 naechsten Bloecke enthaelt (indirekt). Fuer noch groessere Files werden die beiden naechsten Blockadressen fuer eine 2- bzw. 3- fach

indirekte Adressierung verwendet. Kleine Files also im Durchschnitt geringere Zugriffszeiten als groessere (diese Bevorzugung kleiner Elementen zieht sich durch das gesamte System, z.B. tritt sie auch in der Prozessverwaltung auf).

Das bisher Gesagte gilt fuer regulaere Files. Wird eine E/A-Operation fuer ein File ausgefuehrt, dessen i-node dieses File als Spezial-File ausweist, sind die letzten 12 Blockadressen bedeutungslos. Die erste Blockadresse enthaelt dann einen internen Geraetenamen, der als ein Zahlenpaar interpretiert wird, welches Geraetetyp und Geraetenummer angibt. Durch den Geraetetyp ist die Systemroutine festgelegt, die die E/A-Operation fuer diese Geraete behandelt, waehrend durch die Geraetenummer z.B. das entsprechende Laufwerk ausgewaehlt wird.

In dieser Umgebung ist die Implementierung des mount - Systemrufs (Eingliedern eines Filesystems) sehr einfach. Vom System wird dafuer eine Tabelle verwaltet, in welche die i-number und der Geraetename des im Systemruf angegebenen Directorys eingetragen wird. Dazu kommt noch der Name des Special-Files fuer das eingegliederte Geraet. Beim open bzw. create fuer ein File wird diese Tabelle beim Umwandeln des Pfadnames in ein i-number-Geraete-Paar beruecksichtigt. Wird das i-number - Geraete-Paar in dieser Tabelle gefunden, wird einfach mit der i-number des Root-Directories des eingegliederten Filesystems weitergearbeitet. Fuer den Nutzer hat es den Anschein, als wuerden alle E/A-Operationen synchron und ungepuffert ausgefuehrt, da direkt nach einem read - Systemruf die Daten zur Verfuegung stehen und nach einem write - Systemruf der Arbeitsbereich eines Nutzers sofort wieder verwendet werden kann. In Wirklichkeit wird aber vom System durch das bereits erwaehnte blockorientierte E/A-System ein komplizierter Puffermechanismus verwaltet, der die Anzahl von E/A-Operationen bei der Arbeit mit einem File stark reduziert. Nehmen wir an, durch einen write - Systemruf soll ein einzelnes Byte in ein File geschrieben werden, dann ueberprueft das System zunaechst, ob der betreffende Diskettenblock sich im Hauptspeicher befindet. Ist dies nicht der Fall, wird der entsprechende Block eingelesen. Dann wird dort (im Hauptspeicher) das betreffende Byte geaendert und es erfolgt ein Vermerk hierueber in der Blockliste. Anschliessend erfolgt sofort die Rueckkehr vom write -Systemruf, obwohl die eigentliche E/A-Operation noch nicht abgeschlossen ist. Genauso wird beim read - Ruf verfahren. Befindet sich der betreffende Diskettenblock in einem der Systempuffer, so kann das Byte direkt von dort geholt werden und der Systemruf ist abgeschlossen. Im anderen Fall muss zunaechst der betreffende Block in einen Systempuffer gebracht werden, aus dem dann das gewuenschte Byte ausgewaehlt wird. Bei der Arbeit mit dem Filesystem wird stets der Pufferbereich im Hauptspeicher verwendet. Fuer bestimmte Arbeiten (z.B. Diskettenkopierung) ist ein Verwenden des Pufferbereiches nicht sinnvoll. Hierzu stehen auch fuer alle blockstrukturierten E/A-Geraete Spezial-Files zur Verfuegung, die eine Arbeit mit dem Datentraeger ausserhalb der Filesystemverwaltung ermoeglichen.
Dies ist im allgemeinen nur fuer privilegierte Nutzer sinnvoll.

4.3. Prozesse

Prozesse sind in MUTOS 8000 die Umgebung fuer die Programmausfuehrung. Jeder Prozess kann ein oder mehrere Programme (nacheinander) ausfuehren. Durch das Prozessmanagement wird fuer die Zuordnung der Prozesse zu den einzelnen Systemressourcen gesorgt. Dabei wird fuer die Zentraleinheit nach dem Zeitscheibenprinzip verfahren. Durch Systemrufe koennen Prozesse geschaffen oder vernichtet oder einzelne Prozessbestandteile veraendert bzw. ergaenzt werden (z.B. Ausfuehren eines anderen Programms, Oeffnen oder Schliessen von Files u.a.m.). Ausserdem existieren Moeglichkeiten der Prozesssynchronisation.

Bei der Arbeit im Dialog wird beim Aufruf eines Dienstprogrammes vom Kommandointerpreter (Shell) ein Systemruf abgearbeitet, der einen neuen Prozess eroeffnet. Anschliessend wird dieser neue Prozess mit der Ausfuehrung des aktivierten Dienstprogrammes beauftragt. Diese Moeglichkeit der Prozesserstellung und Programmzuordnung kann aber nicht nur vom Kommandointerpreter, sondern auch von allen anderen System- und Anwenderprogrammen genutzt werden.

Im folgenden sollen einige Systemrufe zur Arbeit mit Prozessen kurz erlaeutert werden. Dazu wird dieselbe Darstellung wie im Kapitel zu den Systemrufen des Filesystems benutzt. Diese Schreibweise gleicht weitgehend der Darstellung in C, wo Systemrufe syntaktisch den Funktionsaufrufen entsprechen.

Systemrufe zur Arbeit mit Prozessen

Der Systemruf fork() - (Erzeugen eines neuen Prozesses)

Die einzige Moeglichkeit, einen Nutzerprozess neu zu schaffen, ist der Systemruf

```
processid = fork().
```

Durch diesen Ruf wird der ausfuehrende Prozess in zwei voneinander unabhaengige Prozesse aufgespalten. Beide Prozesse fuehren zunaechst dasselbe Programm (aber voneinander unabhaengig) aus und haben eine gleiche Prozessumgebung (z.B. offene Files). Der neue Prozess (Child) unterscheidet sich vom Originalprozess (Parent) nur durch den Rueckkehrwert (Wert, der einem Funktionsaufruf beim Abarbeiten zugeordnet wird) von fork. Waehrend fork im Parent-Prozess mit dem Prozessidentifikator des neu erzeugten Child-Prozesses zurueckkehrt, ist der Rueckkehrwert im Child-Prozess gleich Null.

Der Systemruf execl() - (Ausfuehren eines Programms)

Ein weiterer wichtiger Systemruf ist

```
execl(file,arg1, ... ,argn,0)
```

Durch diesen Ruf wird der ausfuehrende Prozess veranlasst, den Inhalt des im Argument file angegebenen Files als Programm auszufuehren. arg1 , ..., argn sind Argumente des auszufuehrenden Programms.
Voraussetzung ist natuerlich, dass file auf ein File verweist, welches ein uebersetztes und verbundenes (ausfuehrbares) Programm enthaelt. Bei fehlerhafter Ausfuehrung des execl -Rufes ist ein entsprechender Rueckkehrcode abfragbar.

Wurde execl fehlerlos abgearbeitet,ist das Programm, welches bisher von diesem Prozess abgearbeitet wurde, beendet und durch diesen Prozess nicht mehr erreichbar. Von dem Prozess wird nun als Programm der Inhalt von file abgearbeitet.

Uebliche Verfahrensweise in MUTOS 8000 ist es, zunaechst durch fork einen neuen Prozess zu erzeugen. Beide Prozesse (Parent und Child) fuehren nun dasselbe Programm (voneinander unabhaengig) aus. Im Child-Prozess (processid = 0) kann durch execl dann die Ausfuehrung eines beliebigen anderen Programms veranlasst werden.

Diese Verfahrensweise hat den Vorteil, dass beide Prozesse zunaechst ueber die gleiche Umgebung verfuegen und dort auch Manipulationen, z.B. zur Interprozesskommunikation vornehmen koennen. Andere Systemrufe, die das gleiche leisten, sich aber in der Argumentangabe unterscheiden, heissen execv, execle und execve. Zusammen mit execl werden diese Systemrufe als exec-Systemruf bezeichnet.

Der Systemruf wait() - (Prozesssynchronisation)

Um Prozesse synchronisieren zu koennen, existiert folgender Systemruf:

```
processid = wait(status)
```

Der Prozess, der diesen Ruf ausfuehrt, wird in einen Wartezustand versetzt, bis einer seiner Child-Prozesse die Arbeit beendet. Tritt dies ein, so ist der Rueckkehrwert von wait der Prozessidentifikator des Child-Prozesses, der beendet wurde. Existiert kein Child-Prozess, so wird ein Fehler angezeigt. Durch status werden genauere Informationen ueber den Zustand des Child-Prozesses bei seiner Beendigung gegeben.

exit() - (Beenden eines Prozesses)

Der Systemruf

```
exit(status)
```

beendet die Existenz eines Prozesses und vernichtet alle im Zusammenhang damit existierenden Informationen. Befindet sich der Parent-Prozess im wait -Zustand (wait -Systemruf), wird dieser ueber das Ende der Existenz des Child-Prozesses informiert. Der Wert von status wird an den wait -Ruf des Parent-Prozesses uebergeben.

Weitere Informationen ueber die Arbeit mit Prozessen koennen dem Systemhandbuch-Teil I, Systemrufe (2), entnommen werden.

5. Die Kommandosprache Shell

5.1. Einfuehrung

Die Kommunikation der meisten Nutzer mit dem Betriebssystem MUTOS 8000 erfolgt ueber die Kommandosprache Shell. Der diese Sprache verarbeitende Kommandointerpreter hat keinen besonderen Status innerhalb der Bestandteile von MUTOS 8000. Er ist ein ausfuehrbares Programm, das eine Nutzerschnittstelle zur interaktiven Arbeit realisiert und auslagerbar wie alle anderen Dienstprogramme.
Der volle Umfang der Kommandosprache ist relativ gross. Dafuer bietet Shell aber vielfaeltige Moeglichkeiten, um dem Nutzer die Arbeit unter dem Betriebssystem MUTOS 8000 zu erleichtern.
Shell realisiert u.a.:

- verschiedene Elemente, wie sie aus algorithmischen Sprachen bekannt sind:
  - Programmablauf-Steuerkonstrukte (for,case,if,while)
  - Variablen
  - Parameteruebergabe
- Kommando- und Parametersubstitution
- Filenamen-Generierung
- Umlagerung der E/A-Richtungen
- Modifikation der Umgebung, in der ein Prozess ablaeuft
- Signalbehandlung
- Datenuebergabe zwischen verschiedenen Prozessen ueber Pipes
- Suchen von Files entlang vom Nutzer definierter Wege innerhalb des Filesystems von MUTOS 8000.

Die Shell-Syntax zeichnet sich durch eine knappe Notation aus, aufgebaut aus leicht verstaendlichen und einpraegsamen Zeichen.
Unter Ausnutzung der anwenderfreundlichen Eigenschaften von MUTOS 8000, wie der Transparenz des Filesystems, der vielen Dienstprogramme, die vom Standard-Eingabefile stdin lesen und zum Standard-Ausgabefile stdout ausgeben sowie des Pipe-Konzeptes, bietet Shell dem Nutzer bei wenig Eingabearbeit viel an Systemleistung. Dabei unterstuetzt der Kommandointerpreter sowohl die interaktive Arbeit, d.h. die Eingabe einzelner Kommandozeilen ueber das Bildschirmterminal, als auch das Abarbeiten von Shell-Prozeduren, d.h. von Files, die Shell-Kommandos enthalten. Beide Verarbeitungsarten werden im wesentlichen gleich behandelt.
Das Erlernen der Shell-Programmierung wird dem Nutzer dadurch erleichtert, dass es zu Beginn voellig genuegt, einige wenige Elemente der Kommandosprache und die Wirkungsweise einiger Dienstprogramme von MUTOS 8000 zu kennen, und spaeter bei entsprechendem "Bedarf" nach und nach weitere Moeglichkeiten von Shell hinzuzulernen.

5.2. Einige Moeglichkeiten der Shell-Programmierung

Einfache Kommandos

In der einfachsten Form besteht ein Kommando aus einer Folge von Worten, die durch Leerzeichen getrennt sind. Das erste dieser Worte ist der "Name" des Kommandos. Im Allgemeinen ist dies der Name eines Files, welches ein ausfuerbares Programm enthaelt.
Dieses Programm wird als Ergebnis der Interpretation des angegebenen Kommandos abgearbeitet. Die verbleibenden Worte aus dem Kommando werden durch Shell als Argumente an das abzuarbeitende Programm weitergegeben. Abgeschlossen wird ein Kommando durch das Ende der Eingabezeile oder mit einem `;', wodurch mehrere Kommandos in einer Zeile angegeben werden koennen.
So wird nach Eingabe der Kommandozeile

```
comname arg1 arg2 ... argn
```

ein File mit dem Namen comname gesucht und ueber einen exec-Systemruf dessen Ausfuehrung eingeleitet, wobei die Argumente arg1,arg2,..., argn als Parameter des exec-Systemrufes an das auszufuehrende Programm uebergeben werden.

Hintergrund-Kommandos

Zur Ausfuehrung eines Kommandos wird von Shell i.a. ein neuer Prozess eroeffnet. In diesem neuen Prozess erfolgt das Abarbeiten des aufgerufenen Kommandos, waehrend sich der Prozess, in welchem die Arbeit des Kommandointerpreters ablaeuft, in einem Wartezustand befindet. Dieses Warten dauert solange, bis das Abarbeiten des aufgerufenen Kommandos beendet ist. Waehrend dieser Zeit ist fuer den Nutzer keine Interaktion mit Shell moeglich. Wird ein Kommando jedoch mit dem Operator `&' abgeschlossen, so ist der Wartemechanismus fuer dieses "Hintergrund-Kommando" ausser Kraft gesetzt und die interaktive Arbeit kann unmittelbar nach dem Absetzen des Kommandos fortgefuehrt werden. Die Eingabe der Kommandozeile

```
cc source.c &
```

bewirkt damit zum Beispiel, dass der Nutzer waehrend der Uebersetzung des C-Programms source.c andere Arbeiten an seinem Bildschirmterminal erledigen kann.

Umlagerung der E/A-Richtungen

Die meisten Dienstprogramme von MUTOS 8000 sind so beschaffen, dass sie Eingaben von einem als Standard-Input (stdin) bezeichneten File lesen und ihre Ausgaben zu einem als Standard-Output (stdout) bezeichneten File richten. Fuer einen Nutzer, der unter Shell arbeitet, werden stdin und stdout zu Beginn der Arbeit dem Bildschirmterminal zu geordnet (Spezial-File **/dev/console**), so dass die interaktive Arbeit moeglich ist. Um Ein-/Ausgaben von/nach einem anderen Geraet als dem Terminal bzw. zu irgendeinem anderen File aus dem Filesystem von MUTOS 8000 zu realisieren, kann mit Hilfe der Umlagerungszeichen '<' und '>' eine Aenderung der E/A-Richtung fuer die Dauer der Abarbeitung eines Kommandos erreicht werden. Die folgenden Kommandos bewirken zum Beispiel:

```
pr text        (ohne Umlagerung) Ausgabe des Files text
               auf Bildschirmterminal

pr text >/dev/lp

               Ausgabe von text
               auf Zeilendrucker (Spezial-File /dev/lp )
```

Analog dazu wird die Eingaberichtung mit '<' umgelagert.

Pipelines

Das Bereitstellen des Pipe-Mechanismus durch MUTOS 8000 und die einfache Handhabung, die Shell dazu bietet, ermoeglichen auf unkomplizierte Weise die Kombination von Dienstprogrammen aus MUTOS 8000 untereinander und mit Nutzerprogrammen. Kommandos, die durch den Operator '|' verknuepft sind, wie

```
anycommand | sort | pr
```

werden als Pipeline bezeichnet. Ein derartiges Verknuepfen bewirkt, dass die Kommandos quasiparallel abgearbeitet werden, wobei der Standard-Output jedes Kommandos als Standard-Input des folgenden Kommandos benutzt wird. Damit realisiert die oben angegebene Kommandozeile, dass die Ausgaben des (Nutzer-) Programms anycommand mit Hilfe des Dienstprogramms sort sortiert und anschliessend entsprechend pr formatiert ausgegeben werden.

Programmablauf-Steuerung

Kommandos liefern bei Beendigung ihrer Arbeit einen Rueckkehrcode. Dieser dient als Testwert zur Programmablauf-Steuerung in den Konstruktionen

```
if Kommandoliste1
then Kommandoliste2
else Kommandoliste3
fi
```

und

```
while Kommandoliste1
do Kommandoliste2
done
```

Die genaue Arbeitsweise dieser Konstruktionen soll hier nicht erlaeutert werden. Sie ist jedoch analog zu aehnlichen Formen in hoeheren Programmiersprachen.
Daten- bzw. zeichenkettengesteuerte Verzweigungs- bzw. Schleifenmechanismen werden durch die Konstrukte

```
case Wort in
      Muster1 ) Kommandoliste1 ;;
      Muster2 ) Kommandoliste2 ;;
          .
          .
          .
esac
```

und

```
for Name in Wort1 Wort2 ...
do Kommandoliste
done
```

bereitgestellt. Beim **case**-Konstrukt wird die Zeichenkette Wort nacheinander mit den Mustern Muster1, Muster2, ... verglichen und bei festgestellter Uebereinstimmung die zugehoerige Kommandoliste abgearbeitet. Im **for**-Konstrukt wird die Shell-Variable Name nacheinander mit den Werten Wort1, Wort2, ... belegt und fuer jede Belegung die Kommandoliste abgearbeitet. Mit Hilfe dieser Steuerkommandofolgen kann auf unterschiedliche Weise der Programmablauf in Shell-Prozeduren beeinflusst werden. Eine genauere Beschreibung der Ablaeufe in diesen Folgen, wie auch anderer hier nur angedeuteter sowie vieler weiterer Moeglichkeiten von Shell wird im Systemhandbuch Teil I, Kommandos (1) und ausfuehrlich im Systemhandbuch Teil II, Abschnitt 1 - Kommandointerpreter Shell gegeben.

Shell-Prozeduren

Der Kommandointerpreter Shell kann selbst in einem Kommando aufgerufen werden. Damit besteht die Moeglichkeit , (nicht-interaktiv) Kommandos abzuarbeiten, die in einem File, einer sogenannten Shell-Prozedur, enthalten sind. Ein Aufruf der Form

```
sh procname arg1 arg2 ...
```

bewirkt, dass die Kommandos, die in der Shell-Prozedur procname enthalten sind, nacheinander gelesen und ausgefuehrt werden. Die Argumente arg1, arg2, ... sind dabei in der angegebenen Reihenfolge den Stellungsparametern *1, *2, ... zugeordnet und unter diesen Namen in der Shell-Prozedur verfuegbar. Das Kommando sh kann unter Wirkung einer ganzen Reihe von Optionen ausgefuehrt werden, die z.B. Mechanismen zur Kontrolle der Abarbeitung von Shell-Prozeduren bereitstellen.
Falls dem File procname das Attribut "ausfuehrbar" zugewiesen wurde (mit dem Kommando chmod), kann diese Shell-Prozedur ohne Aufruf des Kommandos sh abgearbeitet werden. Die Kommandozeile hat dann einfach die Gestalt

```
procname arg1 arg2 ...
```

d.h., wie ein "normaler" Kommandoaufruf.
Shell-Prozeduren sind ein wirksames Mittel zum Vereinfachen der Arbeit am Rechner. Da sie keiner Compilation bedueerfen, sind sie leicht anzulegen und zu verwalten. Mit Hilfe der Shell-Optionen wird dem Nutzer das Testen seiner Prozeduren erleichtert.
Der Standard-Input und der Standard-Output einer Shell-Prozedur bleiben waehrend des Abarbeitens dieser Prozedur unveraendert.
Damit koennen solche Prozeduren in Pipelines benutzt werden.

Shell-Variable

Sowohl in Prozeduren als auch waehrend der interaktiven Arbeit koennen Zeichenketten als Werte an Shell-Variable zugewiesen werden. Auf diese Weise laesst sich z.B. das Schreiben haeufig benutzter Namen verkuerzen. So weist die Eingabezeile

```
source=/usr/src/sys/cmd
```

der Shell-Variablen source den Wert **/usr/src/sys/cmd** zu. In einem folgenden Kommando kann diese Zeichenkette unter dem Namen *source erreicht werden.

Shell-Variable koennen mit speziellen Shell-Kommandos an nachfolgend auszufuehrende Prozeduren "exportiert" werden, wie in

export source

oder auch fuer die Dauer einer Prozedur vor Veraenderung geschuetzt werden, wie durch

readonly source

Zum Vereinfachen der Arbeit mit den Shell-Variablen existieren Operatoren (-, =, ?, +), die das Testen und Setzen von Variablen verbinden sowie spezielle Variablen, die durch Shell bereits mit bestimmten Werten belegt sind und vom Nutzer bei seiner Arbeit verwendet werden koennen.

Kommandosubstitution

Eine Zeichenkette, die in Akzentzeichen (`...`) eingeschlossen ist, wird als Kommando betrachtet, das vor Abarbeiten der gesamten zugehoerigen Kommandozeile ausgefuehrt und durch seine nach Standard-Output gerichtete Ausgabe ersetzt werden soll. Das Kommando pwd zum Beispiel liefert als Ausgabe auf Standard-Output den Namen des aktuellen Directory. Wenn nun **/usr/wrk/th** das aktuelle Directory eines Nutzers ist, so weist die Kommandozeile

curdir=`pwd`

der Shell-Variablen curdir den Wert **/usr/wrk/th** zu. Die Kommandosubstitution schafft damit die Moeglichkeit, Programme, die Zeichenketten verarbeiten, unkompliziert in Shell-Prozeduren einzubeziehen. Durch die Kommandosprache selbst werden das Zusammenfuegen von Zeichenketten und das Mustererkennen und -ausfuellen im Zusammenhang mit der Filenamen-Generierung realisiert.

Filenamen-Generierung

Wird ein Filename innerhalb eines Kommandos nicht vollstaendig, sondern in Form eines Musters, gebildet unter Benutzung der Filenamen-Erweiterungszeichen `*', `?', `[...]', angegeben, so ermittelt Shell alle dem Muster entsprechenden Filenamen und setzt diese, alphabetisch geordnet, als einzelne Argumente anstelle des Musters in die Kommandozeile ein. Das Zeichen `*' steht dabei fuer jede beliebige Zeichenkette, `?' fuer ein beliebiges Zeichen und anstelle von `[...]' kann im zu ermittelnden Filenamen irgendeines der in den eckigen Klammern angegebenen Zeichen auftreten.

Daher listet das Kommando

```
ls *.c
```

alle Filenamen im aktuellen Directory auf, die auf ".c" enden und

```
ls /dev/rk?
```

liefert die Namen aller Diskettenlaufwerke (rk0, ..., rk3), fuer die in **/dev** ein Eintrag vorhanden ist.

5.3. Implementierung

Ein vom Nutzer am Bediengeraet eingegebenes Kommando bzw. ein aus einer Shell-Prozedur eingelesenes Kommando wird von Shell zunaechst auf seine syntaktische Richtigkeit geprueft. Werden keine Fehler festgestellt, so erfolgt die Ausfuehrung aller angegebenen Substitutionen (Parametersubstitution, Kommandosubstitution, Filenamen-Generierung) und das Zerlegen des Kommandos in seine einzelnen Bestandteile. Damit hat das Kommando die fuer seine spaeter erfolgende Abarbeitung ueber einen exec-Systemruf noetige Form erhalten.

An dieser Stelle wird von Shell durch einen fork-Systemruf ein neuer Prozess eroeffnet. Dieser Child-Prozess "erbt" von seinem Parent-Prozess (der urspruenglich arbeitenden Shell) die geoeffneten Files, d.h. mindestens diejenigen, die als Standard-Input und Standard-Output benutzt werden, und arbeitet asynchron mit ihm weiter. Ausser beim Abarbeiten eines Hintergrund-Kommandos wartet der Parent-Prozess (Shell) auf das Ende des gestarteten Child-Prozesses, d.h. auf die Beendigung der veranlassten Kommandoabarbeitung. Im Child-Prozess werden zunaechst die ggf. erforderlichen E/A-Umlagerungen durchgefuehrt, wie in 4.2 beschrieben. Im Parent-Prozess bleibt der alte Zustand bzgl. der E/A-Richtungen erhalten! Danach wird ueber einen exec-Systemruf das Abarbeiten des angegebenen Kommandos gestartet. Aus einem erfolgreichen exec-Ruf gibt es keine Rueckkehr in den diesen Ruf sendenden Prozess. Der Child-Prozess wird daher mit Abschluss der Kommandoabarbeitung beendet, und die Arbeit von Shell wird (im Parent-Prozess) fortgesetzt. Shell bekundet durch Anzeige der Eingabeanforderung (Promptzeichen, i.a. '$') auf dem Bildschirm seine Bereitschaft, ein neues Kommando einzulesen bzw. arbeitet das folgende Kommando in einer Shell-Prozedur ab.
Der Haupt-Shell-Prozess eines Nutzers, d.h. der Shell-Prozess, ueber den der Nutzer zu Beginn seiner Arbeit am Terminal die Kommunikaiton mit MUTOS 8000 aufnimmt, ist selbst Child eines anderen Prozesses. Dieser andere Prozess wurde als letzter Schritt waehrend der Initialisierung des Systems gestartet und arbeitet das Programm init ab. Durch dieses Programm wird ein Prozess eroeffnet, das Terminal fuer Ein- und Ausgaben ueber die Filedeskriptoren 0, 1 und 2 geoeffnet, und die Login-Prozedur gestartet.

Nach erfolgreichem Eintragen des Nutzers ins laufende System (login) wird in dessen Home-Directory, ein im Schutzwort-File (**passwd**) fuer jeden Nutzer festzulegendes Directory, verzweigt.
Der Nutzer-Identifikationscode (Nutzer-ID) des laufenden Prozesses wird gleich dem Nutzer-ID des Nutzers, der sich eingetragen hat, und die Abarbeitung von Shell wird gestartet.
Shell arbeitet zuerst, falls vorhanden, die Shell-Prozedur **.profile** im Home-Directory des Nutzers ab. Dort koennen Kommandos zusammengefasst sein, von denen der Nutzer moechte, dass sie jeweils vor Beginn seiner Arbeit unter MUTOS 8000 ausgefuehrt werden. Danach ist Shell bereit, Eingaben des Nutzers zu interpretieren.
Dieser, wie hier beschrieben initialisierte, Haupt-Shell-Prozess eines Nutzers befindet sich die meiste Zeit im Zustand des Wartens auf Eingaben ueber das Terminal.

6. Uebersicht ueber Systemprogramme

MUTOS 8000 stellt eine Vielzahl von Dienstleistungen ueber Systemprogramme (Dienstprogramme) zur Verfuegung. Diese existieren getrennt vom MUTOS-Systemkern, demgemaess sind unterschiedliche Ausbaustufen moeglich. Entsprechend der MUTOS-8000-Systemphilosophie existieren dabei keine grossen Programme mit vielen Dienstleistungen, sondern eine Vielzahl kleinerer Programme mit begrenztem, genau umrissenen Leistungsumfang. Die Flexibilitaet insbesondere gegenueber Modifizierungen und Ergaenzungen ist dadurch sehr hoch. Die Dienstprogramme sind ueber entsprechende Kommandos aufzurufen. Beim Aufruf eines solchen Dienstprogramms koennen durch entsprechende Optionen bzw. bei einigen ganz wenigen Dienstprogrammen (z.B. dem Editor) auch durch weitere Kommandos, die geforderten Dienste genauer klassifiziert werden. Bei der Ausfuehrung der Kommandos kommt dem Kommandointerpreter Shell (Abschnitt 4) eine wichtige Rolle zu.
Die Systemprogramme lassen sich entsprechend den geleisteten Diensten in verschiedene Gruppen einteilen. Hierzu gehoeren Programme:

- zur Steuerung des Nutzerzugriffs
- zum Einstellen der Terminalfunktionen
- zur Filemanipulation
- zur Manipulation von Directories und Filenamen
- zum Abarbeiten von Nutzerprogrammen
- fuer Statusabfragen
- fuer Wartung und Betrieb des Systems
- zur Programmentwicklung (Programmentwicklungswerkzeuge)
- zur automatischen Textverarbeitung

Im folgenden werden die entsprechenden Dienstprogramme, nach obigen Gruppen gegliedert, mit einer Kurzinformation ueber die von ihnen geleisteten Dienste aufgefuehrt.

Dienstprogramme zur Steuerung des Nutzerzugriffs

login
bewirkt das Anmelden eines Nutzers im Betriebssystem. Das Anmelden erfolgt durch Eingabe eines Namens, unter dem das System den jeweiligen Nutzer "kennt". Das System ueberprueft, ob ein Nutzer mit diesem Namen existiert und ob sein Directory eventuell durch ein Schutzwort (siehe Dienstprogramm **passwd** unten) gesichert ist.

passwd
dient zum Aendern bzw. Installieren eines Schutzwortes ("password"). Jeder Nutzer kann beim Anmelden (siehe **login**) den Zugang zu seinem eigenen Directory zusaetzlich durch ein von ihm selbst gewaehltes Schutzwort sichern. Entsprechendes gilt auch fuer das System selbst.

Dienstprogramm zur Einstellung der Terminalfunktionen

stty
dient dem Einstellen von fuer die jeweilige Aufgabe optimalen Terminalparametern.

Dienstprogramme zur Manipulation von Files

cp
bewirkt das Kopieren von Files unabhaengig von Typ und Inhalt.

cmp
vergleicht zwei Files miteinander und teilt mit, ob Unterschiede existieren.

cat
dient zum Verketten beliebiger Files. Das resultierende File wird normalerweise auf dem Standardausgabegeraet ausgegeben, wenn nicht durch Spezifikation andere Verfuegungen getroffen wurden. Das Dienstprogramm **cat** ist somit auch zur Ausgabe von einzelnen Files z.B. auf dem Terminal geeignet, wobei die Ausgabe ohne jegliche Zusatzangaben (Kopfdruck, Seitenvorschub u. dgl.) erfolgt.

pr
bewirkt den formatierten Ausdruck von Files mit Titelangabe, Datum und Seitenzaehlung auf jeder Seite. Moeglich ist auch mehrspaltiger Ausdruck.

dd
dient zum Uebertragen von Files und zum Konvertieren von Fileformaten.

Dienstprogramme zur Manipulation von Directories (Verzeichnissen) und Dateinamen

crm
loescht einzelne Files und ganze Directory-Hierarchien.

ln
vergibt Aliasnamen an Files (Erzeugen von "links").

mv
ermoeglicht das Umbenennen (und gegebenenfalls Transportieren) von Files.

chmod
dient dem Aendern von Zugriffsrechten zu Files und Directories.

chown
aendert den Eigentuemer eines oder mehrerer Files.

mkdir/rmdir
bewirken das Anlegen eines neuen bzw. das Loeschen eines vorhandenen Directories.

cd
bewirkt das Aendern des aktuellen Directorys eines Nutzers.

pwd
gibt das aktuelle Directory aus.

Dienstprogramme zum Abarbeiten von Nutzerprogrammen

sh
dient zum Aktivieren des Kommandointerpreters Shell, der wesentliche Dienste zum Ausfuehren von Nutzerprogrammen leistet
- weitere Informationen siehe Abschnitt 5.

echo
gibt seine Argumente auf das Standardausgabegeraet aus (spezifizierbar sind auch andere Moeglichkeiten). Das Systemprogramm **echo** kann z.B. zur Ausgabe von Diagnoseinformationen in Shell-Prozeduren verwendet werden.

kill
beendet den spezifizierten Prozess.

Dienstprogramme fuer Statusabfragen

ls
dient dem Auflisten von Filenamen aus vorgegebenen Directories mit entsprechend spezifizierbaren Zusatzinformationen.

date
gibt das aktuelle Datum und die Uhrzeit aus. Das Kommando ermoeglicht auch die Eingabe der obigen Werte.

df
gibt die Anzahl der freien Bloecke des spezifizierten File-Systems bzw. aller eingegliederten File-Systeme aus.

who
gibt den System-Namen des zur Zeit im System angemeldeten Nutzers (siehe **login**) und die Zeit seiner Anmeldung aus.

ps
dient der Ausgabe von Informationen ueber momentan laufende Prozesse.

pstat
dient der Ausgabe von Systeminformationen, indem es den Inhalt von Systemtabellen interpretiert.

pwd
gibt den Namen der fuer den anfragenden Nutzer aktuellen Directories aus.

Dienstprogramme fuer Wartung und Betrieb des Systems

mount/umount
dient dem Eingliedern (bzw. Entfernen) von Disketten mit vollstaendigem File-System an den existierenden Directory-Baum.

mkfs
dient zum Einrichten eines neuen File-Systems auf dem spezifizierten Geraet.

mknod
dient zum Einrichten eines File-System-Eintrittspunkts ("i-node") fuer ein Spezial-File. (Spezial-Files sind entsprechend der Systemphilosophie physische Geraete, virtuelle Geraete, physische Speicher usw.)

dcheck/icheck/ncheck
dienen zur Kontrolle des File-Systems. Dabei werden entsprechende Ausgabeinformationen generiert. Das Wiedergewinnen nicht ordnungsgemaess erfasster Bloecke ist (Reparatur des File-Systems) moeglich.

clri

entfernt fehlerhaft belegte Bloecke aus dem File-System und ermoeglicht im Zusammenwirken mit den **check**-Dienstprogrammen (siehe oben) Reparaturen des File-Systems.

sync

veranlasst, dass alle noch ausstehenden E/A-Aktivitaeten des Systems zu Ende gebracht werden und das File-System auf der Diskette vom Hauptspeicher aus aktualisiert wird (Herausschreiben des Super-Blocks).
Das Dienstprogramm ist zu benutzen, um bei Beendigung der Arbeit mit MUTOS 8000 einen ordnungsgemaessen Abschluss zu erzielen.

Grundlegende Dienstprogramme zur Programmentwicklung (Programmentwicklungswerkzeuge)

ar

dient zur Pflege von Filegruppen, die aus Effizienzgruenden zu Archiven bzw. Bibliotheken zusammengefasst sind.

as

ist der PLZ/ASM-Assembler des Systems. Im System MUTOS 8000 besitzt der Assembler im Vergleich zu anderen Systemen eine aeusserst untergeordnete Rolle. Seine Aufgaben werden durch die Programmiersprache C wahrgenommen (Abschnitt 7).

adb

ist ein interaktives Testsystem. U.a. sind folgende Leistungen moeglich:
- Interaktives Testen mit Unterbrechungspunkten
- Postmortem-Dumps (Speicherabzuege nach Programmabbruch)
- Suche nach vorgegebenen Mustern
- Symbolische Bezugnahmen auf lokale und globale Variable

cc

aktiviert den Compiler fuer die Programmiersprache C.
Weitere Informationen dazu siehe Abschnitt 7.

od

ermoeglicht die zeichenweise Ausgabe beliebiger Files. Je nach Spezifikation koennen die Zeichen mit ihrem Oktalwert, als Zeichendarstellung oder hexadezimal wiedergegeben werden. Eine Moeglichkeit ist z.B. die Kontrolle auf nichtdruckbare Zeichen.

ld

ist der Lader fuer Objektprogramme. Er bietet die Moeglichkeit, verschiedene Objekt-Files zu verbinden und die erforderlichen Routinen aus Bibliotheken einzufuegen. Der entstehende Modul kann entweder ausgefuehrt oder "aufgehoben" werden.

nm
ermoeglicht die Ausgabe der Symboltabelle (Namensliste) von Objektprogrammen.

size
gibt die Speicherplatzanforderungen der spezifizierten Objekt-Files aus.

strip
dient zum Entfernen von Symboltabelle und Verschiebeinformationen aus dem Objektcode, die normalerweise zu den Ausgabe-File von Assembler und Lader hinzugefuegt werden. Nach dem Testen eines Programms kann so Platz gespart werden.

Programme zur automatisierten Textverarbeitung

ed
ist der interaktive Texteditor. Er ist ein universelles Textbearbeitungsprogramm fuer beliebigen Eingangstext und dient auch zur Programmeingabe und -aufbereitung.

nroff
ist ein Textformatierungsprogramm. Es ermoeglicht ein komfortables Erstellen von Druckdokumenten auch durch EDV-maessig ungelernte Kraefte. Die Eingabe besteht aus Textzeilen und Makrozeilen zur Steuerung der Textformatierung.
Zur Arbeit mit **nroff** gibt es einen Standard-Makrosatz zum Anlegen von Manuskripten.

Dienstprogramm zum Bearbeiten von Datenmengen

wc
ermoeglicht das Zaehlen von Zeilen, "Worten" (durch Leerzeichen getrennte Zeichenketten) bzw. Zeichen eines Files.

7. Beschreibung der Programmiersprache C

7.1. Uebersicht

C ist eine vielseitig verwendbare Programmiersprache. Sie ist gekennzeichnet durch

- Moeglichkeiten zur Strukturierung von Daten und der Programmablaufsteuerung,
- ein flexibel nutzbares Datentypkonzept,
- eine umfangreiche Menge von Operatoren und
- eine kompakte Schreibweise.

Mit C koennen Programme fuer numerische wie auch nichtnumerische Probleme geschrieben werden.
C ist keine "very high level" Sprache, aber doch eine hoehere Programmiersprache. Mit ihr wurde der erfolgreiche Versuch unternommen, die Vorzuege hoeherer Programmiersprachen mit bewaehrten Prinzipien guter Assemblerprogrammierung zu verbinden. Daher ist es mit C moeglich, auch maschinennahe Programme zu schreiben.
Die Grundeinheiten der Programmiersprache sind

- Definitionen von Daten und Funktionen als deskriptive Bestandteile und
- algorithmische Ausdrucksmittel, naemlich die Anweisungen in den Funktionen, als operative Bestandteile.

Bei der Entwicklung der Sprache C war man bemueht, die Moeglichkeiten gegenwaertig verfuegbarer Rechner in der Sprache zu reflektieren. Einerseits wurde damit eine relativ einfache Implementierung des Compilers gesichert. Andererseits wurde auf diese Weise erreicht, dass die durch den C-Compiler uebersetzten C-Programme hinreichend effizient sind und kein Zwang fuer eine weitere Assemblerprogrammierung besteht. Obwohl C der Leistungsfaehigkeit heutiger Rechner gerecht wird, ist die Sprache unabhaengig von irgendeiner Maschinenarchitektur. Die Allgemeinheit der Ausdrucksmittel von C und das Nichtvorhandensein laestiger Einschraenkungen bewirken eine bequeme, flexible und damit effektive Benutzung der Sprache. Der Programmierer wird durch C nicht wie z.B. in PASCAL zur Disziplin gezwungen. Der Flexibilitaet von C muss sich der Programmierer durch eine strenge Selbstdisziplin wuerdig erweisen.

Ihre Eignung als Systemprogrammiersprache hat C bewiesen, indem das vorliegende Operationssystem MUTOS 8000 zum groessten Teil in C programmiert wurde.

7.2. Programmstruktur

C-Programme bestehen aus einer Sammlung von Definitionen der Variablen und Funktionen.
Ein C-Programm kann physisch in einem oder mehreren Textfiles stehen, die unabhaengig voneinander uebersetzt werden koennen.

Die Funktionen sind in C die einzige Art von Unterprogrammen. Sie enthalten den die Algorithmen beschreibenden Programmtext. In anderen Programmiersprachen werden im allgemeinen zwei Arten von Unterprogrammen unterschieden, z.B. in ALGOL 60 die Verfahrensprozeduren und die Funktionsprozeduren. In C wird die Funktion fuer beide Zwecke eingesetzt.

Argumente werden an Funktionen uebergeben, indem die Werte der Argumente kopiert werden. Die aufgerufene Funktion ist nicht in der Lage, das Argument ausserhalb der Funktion zu veraendern (call by value). Dies stellt aber keinen Nachteil dar. Man kann auf diese Weise Hilfsvariablen einsparen, wie sie sonst bei der von anderen Programmiersprachen gewohnten Parameteruebergabe ueber Adresse (call by reference) erforderlich sind.
Man kann in C als Funktionsargumente auch Zeiger uebergeben und somit eine Parameteruebergabe ueber Adresse realisieren. Die Funktion kann in solch einem Fall das Objekt, auf das der Zeiger zeigt, veraendern, wobei der Zeiger selbst wertmaessig gleich bleibt.

Jede Funktion kann rekursiv aufgerufen werden. Funktionen duerfen nicht verschachtelt definiert werden. Die Funktionen koennen untereinander Daten sowohl ueber ihre Argumente als auch ueber globale Variable austauschen.
C besitzt ein Blockkonzept, das bezueglich der Gueltigkeitsbereiche der Variablen vergleichbar zu dem von ALGOL oder PL/1 ist. Das Innere einer Funktion ist ein Block.
In der Sprache C selbst gibt es keine Anweisungen fuer die Ein- und Ausgaben. Derartige und andere Leistungen des Operationssystems werden in Form von Funktionsaufrufen in den Programmen angefordert. Die erforderlichen Funktionen sind in den Standardbibliotheken enthalten (siehe Systemhandbuch Teil I, Bibliotheksfunktionen (3)). Bezueglich der Schreibweise ist C formatfrei.

7.3. Daten

Werte werden in C durch die von den heute ueblichen Rechnern zur Verfuegung gestellten Datenobjekte Byte, Wort und Mehrfachwort implementiert. Die Werte lassen sich entsprechend ihren Eigenschaften zu bestimmten Wertmengen, den sogenannten Datentypen, zusammenfassen.

In C sind die grundlegenden einfachen Datentypen:

- Zeichen (character),
- ganze Zahl (integer) und
- reelle Zahl (floating point number) sowie
- Aufzaehlungstyp (enumeration).

Die Zahlen koennen mit verschiedenen Genauigkeiten angegeben werden.
Der Aufzaehlungstyp in C entspricht in PASCAL dem Datentyp Scalar. Bei einem Aufzaehlungstyp werden, wie es der Name sagt, die Werte aufgezaehlt. Die aufgezaehlten Werte haben den Charakter von Konstanten.

Aus den einfachen Datentypen koennen Hierarchien von Datentypen konstruiert werden. Es gibt in C folgende hoeheren Datentypen:

- Feld (array),
- Verbund (structure),
- gemeinsamer Bereich (union),
- Funktion (function) und
- Zeiger (pointer).

Jede Variable gehoert zu einem bestimmten Datentyp. Diese Information wird bei der Definition bzw. Deklaration einer Variablen festgelegt.
Es gibt die Moeglichkeit, Variablen zu initialisieren.Ein Feld ist eine zusammengesetzte Datenstruktur, deren Komponenten vom gleichen Datentyp sind. Der Zugriff zu den Komponenten erfolgt ueber Indizes. Felder sind vergleichbar mit ein- oder mehrdimensionalen Matrizen. Das Feld von C gleicht dem Array in ALGOL 60 oder PASCAL bzw. den Reihungen in ALGOL 68.
Ein Verbund fasst Daten verschiedener Datentypen zu einer Einheit zusammen. Der Zugriff zu den Komponenten erfolgt ueber den Namen der Verbund-Variablen und den Komponenten Namen. Diese Datenstruktur ist vergleichbar mit dem Record in PASCAL oder der Structure in PL/1 bzw. in ALGOL 68.
Der Datentyp "gemeinsamer Bereich" ermoeglicht, dass eine Variable zu verschiedenen Zeitpunkten Objekte verschiedenen Datentyps und Speicherplatzbedarfs enthalten kann. Der Compiler sorgt fuer die Groesse und die Ausrichtung des erforderlichen Speicherplatzes. Die gemeinsamen Bereiche von C entsprechen in PASCAL dem varianten Teil des varianten Records.

Eine Variable vom Datentyp Zeiger enthaelt die Adresse einer anderen Variablen oder Funktion.
In C muessen alle Variablen vor ihrer Benutzung definiert sein. Eine Variablendefinition besteht aus dem Namen des Datentyps und einer Liste der Variablen.
Der Gueltigkeitsbereich von Variablen kann mit Hilfe von Schluesselworten programmiert werden. Variable koennen lokal fuer eine Funktion, global innerhalb eines Files und schliesslich global fuer das gesammte C-Programm sein.
Fuer Konstanten koennen symbolische Namen festgelegt und mit diesen Namen kann anstelle der Konstanten operiert werden.

7.4. Steuerstrukturen

In C gibt es zur Strukturierung des Programmablaufs folgende grundlegenden Ausdrucksmittel:

- Block bzw. Anweisungsverbund,
- Zweiwegeauswahl (if),
- Schleifen mit Abbruchtest am Anfang (while, for),
- Schleifen mit Abbruchtest am Ende (do),
- Mehrwegeauswahl (switch) und
- Sprunganweisung (goto, break, continue).

7.5. Operatoren

In C gibt es folgende Klassen von Operatoren:

- arithmetische Operatoren,
- Vergleichsoperatoren,
- bitorientierte Operatoren,
- Zuweisungsoperatoren,
- Entscheidungsoperatoren,
- Zeigeroperatoren.

Mit Hilfe der Operatoren werden die Daten verknuepft. Die Ergebnisse werden anderen Datenobjekten zugewiesen oder beeinflussen die Steuerung des Programmablaufs.
Bei der Verarbeitung von Daten verschiedener Datentypen in einem Ausdruck, z.B. einem arithmetischen, realisiert der C-Compiler teilweise eine Datenumwandlung automatisch. Die Ueberpruefung der Datentypen wird vom C-Compiler nicht so streng gehandhabt wie in PASCAL oder ALGOL 68.
Variable, die von einem zusammengesetzten Datentyp sind, koennen nur komponentenweise verarbeitet werden.

8. Bibliotheken

Im Betriebssystem MUTOS 8000 stehen dem Nutzer zwei Bibliotheken zur Verfuegung, durch deren Benutzung er Zugriff zu saemtlichen Systemrufen und zu einer ganzen Reihe verschiedener Funktionen hat. Die Funktionen mathematischen Inhalts sind in der Bibliothek /lib/libm.a enthalten. Neben trigonometrischen, hyperbolischen, Exponentialfunktionen u.a. sind auch verschiedene Besselfunktionen realisiert.
Die Bibliothek /lib/libc.a beinhaltet die Eintrittspunkte zu den Systemgrundfunktionen (Systemrufe), die dem Nutzer zur Verfuegung stehen. Weiterhin sind in dieser Bibliothek eine ganze Reihe von Funktionen archiviert, die auf den Systemrufen aufbauen und mit mehr Komfort fuer den Anwender verschiedene Verbindungen zu Systeminformationen und -diensten realisieren sowie wirksame Hilfsmittel zur Durchfuehrung der in Nutzerprogrammen vorgesehenen E/A-Arbeit bereitstellen. Kernstueck derletztgenannten Bibliothek ist das Standard-E/A-Paket. Die darin enthaltenen Funktionen und Makros bieten verschiedene Moeglichkeiten, um die zur E/A-Arbeit noetigen Grundfunktionen Oeffnen, Schliessen, Loeschen von Files, Speicherplatzzuweisung, Pufferbehandlung, Positionieren innerhalb Files, Testen moeglicher Fehler sowie Lesen und Schreiben auszufuehren. Das Standard-E/A-Paket zeichnet sich durch die Einfachheit seiner Benutzung aus. Die darin enthaltenen Funktionen sind mit dem Ziel einer moeglichst hohen Zeit- und Speicherplatzeffektivitaet geschrieben und garantieren durch ihre Maschinenunabhaengigkeit die Portabilitaet zu anderen Rechnersystemen.
Die Bibliothek /lib/libc.a wird vom C-Compiler automatisch geladen, kann aber auch, wie die mathematische Bibliothek, durch eine einfache Lader-Option explizit zum Durchsuchen nach benoetigten Funktionen bereitgestellt werden.

Anhang

Erklaerung der Fachbegriffe zur Dokumentation MUTOS 8000

Ampersand	das Zeichen "&"
Archivelementname	File, das in einem Archivfile abgespeichert ist.
Ausgabeschlange	Die Zeichen, die sich im Ausgabepuffer befinden.
Autoimkrementregister	Register, dessen Inhalt durch bestimmte Operationen automatisch erhoeht wird.
Automatic Variable	Speicherklasse der C-Sprache
Backslash	das Zeichen "\"
Backspace	Rueckschritt
Bessel-Funktionen	Amplitueden der Grundwelle und der Oberwellen in Abhaengigkeit des Phasenhubes bei Frequenzmodulation.
Bold	Breitschrift
booten	Das System anlaufen lassen.
Break	Abbruch des Auffuellens einer Zeile (Textformatieren)
BSS-Bereiche	Bereich nicht-initialisierter Daten
Child-Prozess	Der durch einen anderen Prozess (Parent-Prozess) erzeugte Prozess (wortl.: "Kind-Prozess")
Common-Region	Spezieller Speicherbereich fuer Daten im Programmabbild.
Core-File	Speicherabzug fuer spaeteren Test.
C-Backtrace	Liste der aufgerufenen Unterprogramme (Funktionen) zur Programmtestung.
Debugger	Testhilfsprogramm
dekrementieren	Verringern eines Wertes.
Delay	Verzoegerungszeit
Directory	Verzeichnis (von Files)
Diversion	Einstreuen einer Zeichenfolge in einen Datenstrom
Driver	Treiber
Duplex-Betrieb	Parallele Ein- und Ausgabe
Echobetrieb	Anzeige, der ueber Tastatur eingegebenen Daten auf Bildschirm bzw. Ruecksenden der von einem Terminal empfangenen Daten zur Kontrolle.
Eingabeschlange	Zeichen, die sich im Eingabepuffer befinden.
Environment	Umgebung
ESC-Folge)	Folge von Steuerzeichen zur Durckausgabe
Escape-Folge)	
ESC-Zeichen	Zeichen, das eine Folge von Steuerzeichen eingeleitet.
Euklidischer Abstand	Betrag komplexer Zahlen

Feldmodus	Anordnen der Daten in einem festgelegten Feld.
File-Diskriptor	Eine Zahl zwischen 0 und 19, die zur Identifizierung des Files bei Systemrufen und bei Shell-Sprachelementen dient.
File header	Filekennsatz
Filter	Ein Programm, das von Standard-Input liest und nach Standard-Output ausgibt und damit innerhalb eine Pipeline angeordnet werden kann.
Floatzahlen	Gleitkommazahlen
Fuellmodus	Bei Textformatierung die Betriebsart "Auffuellen der Zeile bis zum rechten Rand".
global	gesamt; bei ed(1) Operation, die sich auf die gesamte Zeile bezieht.
Halbduplexes Terminal	Ein-/Ausgabe nicht parallel
Header-file	Spezielles Definitionsfile, das durch den Sufix ".h" gekennzeichnet ist und die Macro-Definitionen beinhaltet
Inch	engl. Laengenmass
Include-file	s. Header-file
inkrementieren	Erhoehen eines Wertes.
i-node	Beschreibung eines Files auf Diskette.
i-node-Nummer	Enspricht der Adresse des i-nodes.
i-node-Eintragung	Element des i-nodes.
Input-Filter	Programm das seine Ausgabedaten an ein anderes Programm ueber eine Pipe gibt.
interaktiv	wechselseitig wirkend; bei MUTOS Arbeit mit der Tastatur.
interleave	verschachteln; bei MUTOS die Reihenfolge der physischen Saetze auf der Diskette.
Interrupt-Signal	eines der MUTOS-Signale (siehe signal(2))
Italic	Schraeg- bzw. Kursivschrift
Link-Zaehler	Anzahl der Bindungen eines Files
Login	Eintragen des Nutzers in das System.
Login-Name	Name, unter dem sich der Nutzer in das System eintraegt.
Logout	Abmelden des Nutzers (durch Quit-Signal)
Macro-Preprozessor	Programm zur Vorverarbeitung von Macros.
Map	Liste
Magic-Number	Wort zum Kennzeichenen bestimmter Files (z.B. a.out-Format, ar-Format).
Major-Geraetenummer	Geraetetyp
Makro	Vorgefertigte Programmteile, die in ein Anwenderprogramm eingebunden werden.
Minor-Geraetenummer	Laufwerksnummer
Modem	Einrichtung fuer Datenfernuebertragung
Modus	Betriebsart; speziell die Zugriffsrechte eines Files.

Modusbits	Die Bits fuer die Zugriffsrechte
Monadische Operatoren	einfache Operatoren
Multiuserbetrieb	Mehrnutzerbetrieb (mehrere Nutzer teilen sich die Resourcen des Systems)
Nicht-Fuell-Modus	Betriebsart, bei der kein Auffuellen der Zeile erfolgt (Textformatierung).
No-Space-Modus	Betriebsart "kein Zwischenraum" (Textformatierung)
Offset	Verschiebung
Option	Zusatz, Moeglichkeit
Output-Filter	Programm, das seine Daten von einem anderen Programm ueber eine Pipe erhaelt.
Overstrike-Funktion	Ueberbruekungsfunktion
Parent-Directory	In der File-Hierarchie das naechst hoehere Directory.
Parent-Prozess	Prozess, der einen oder mehrere andere Prozesse (Child-Prozesse) erzeugt hat (woertlich: Elternprozess).
Password	Kennwort - dient dem Schutz der Daten vor unbefugtem Zugriff.
Pfadname	In der Filesytemhierarchie die Liste der Directories bis zu dem Filenamen oder nur der Filename.
Pica	Entspricht 1/6- inch
Pipe)	Datenfluss zwischen kommuniszierenden
Pipeline)	Prozessen.
Pointer	Zeiger
Polling-Betrieb	Zyklusbetrieb
Positionstraps	Ausloesen einer Ausnahmebehandlung bei Erreichen bestimmter Positionen (Textverarbeitung).
Postprozessor	Programm, das die Ausgabedaten eines anderen Programmes verarbeitet.
Prompt-Zeichen	Bereitschaftszeichen
Prozessidentifikator	Die Zahl, die jedem Prozess zur Identifizierung zugeordnet wird.
"Raw"-Geraet	Ein- und Ausgabe erfolgen direkt ohne Pufferung
Relocations-Bits	Verschieblichkeitsinformationen in Objekt-Files
Roman	Normalschrift
root-Filesystem	Root- (oder Wurzel) Diskette
Rueckkehrwert	Der Wert der von jeden Prozess, Bibliotheksfunktion oder Systemruf an die jeweilige aufrufende Ebene zurueckgegeben wird.
Scheduler-Prozess	Prozess zur Prozessverwaltung
Shell	Kommandosprachenineterpreter (eigentlich Schale; abgeleitet von dem MUTOS-Schalenmodell)
Sequenz	Folge
Stack	Stapel-Speicher, dessen zuerst eingespeicherten Daten zuletzt wieder ausgegeben werden.

File-Pointer	Zeiger auf die Struktur FILE; dient der Identifizierung des Files bei Bibliotheksfunktionen.
Static-Variable	Speicherklasse der C-Sprache
Stellungsparameter	Bei Shell die uebergebenen Argumente
Steuerkonstrukte	Sprachelemente der Shell fuer Verzweigungs- und Schleifenmechanismen.
Sticky Bit	Retten des Textobbildes
Stream	Datenstrom
String	Zeichenkette
Suchpfad	Die Directories in denen Shell nach einem ausfuehrbaren File sucht.
Superblock	der ein Filesystem beschreibende Block
Super-User	Nutzer mit besonderen Privelegien
Swap-bereich	Bereich auf Diskette, der zur Aufnahme ausgelagerter Prozess dient.
Terminal-Interrupt	durch Betaetigen definierter Terminaltasten ausgeloester Unterbrechung
Transparenz	Durchsichtigkeit, Klarheit
Trap	Spezielle Behandlung, die beim Eintreten bestimmter Ereignisse ausgefuehrt wird und eine Unterbrechung der laufenden Behandlung bewirkt.
Vertikalkonstruktion	senkrechter Aufbau einer Seite (Textvormatierung)
Unter-Shell-Prozess	Ein Shell-Prozess, der von einen anderen Shell-Prozess zur Abarbeitung einer Shell-Prozedur ge-startet wurde.
Urlader	Systemlader

Kv 461/86 - V 3/15 - 444