ANWENDER-DOKUMENTATION	Betriebssystem MUTOS 1700	MOS
11/87		MUTOS 1700

Anwendungsbeschreibung Betriebssystem MUTOS 1700

AC A 7100 , A 7150

VEB Robotron-Projekt Dresden

Die vorliegende Dokumentation entspricht dem Stand von 11/87.

Die Ausarbeitung erfolgte durch ein Kollektiv des
VEB Robotron-Elektronik Dresden Stammbetrieb des VEB Kombinat Robotron.

Herausgeber:

VEB Robotron-Projekt Dresden
Leningrader Str. 9
Dresden 8010

Kurzreferat

Die vorliegende Schrift ist eine Einführung in das interaktive Teilnehmersystem MUTOS 1700 (Multi-User-Timesharing-Operating-System) für die Arbeitsplatzcomputer A 7100 und A 7150 des VEB Kombinat Robotron.
Abschnitt 1 erläutert zunächst Einsatzgebiete, Bestandteile und Systemphilosophie des Betriebssystems MUTOS 1700. Im Abschnitt 2 wird die erforderliche Gerätekonfiguration vorgestellt.
Abschnitt 3 bringt eine Darstellung über Aufbau und Arbeitsweise des MUTOS-Systemkerns und des MUTOS-File-Systems. Erläutert werden auch wichtige Systemrufe.
Im Abschnitt 4 wird auf die Kommandosprache Shell und den zugehörigen Kommandointerpreter eingegangen.
Eine Zusammenfassung der unter MUTOS zur Verfügung stehenden Systemdienstprogramme wird im Abschnitt 5 gegeben.
Die Realisierung der Fernkopplungsunterstützung von Rechnern unter MUTOS bzw. zu kompatiblen Systemen wird im Abschnitt 6 beschrieben.
Im Abschnitt 7 wird die höhere Programmiersprache C vorgestellt, die auch als Systemprogrammiersprache fungiert.
Abschnitt 8 geht auf die zur Verfügung stehenden Systembibliotheken ein, und Abschnitt 9 beschreibt die Generierung des Betriebssystems MUTOS 1700.

Inhaltsverzeichnis Seite

1. Aufgabenstellung

G1.1. Allgemeine Gesichtspunkte

Das hier vorgestellte interaktive Teilnehmersystem MUTOS 1700 (Multi-User-Timesharing-Operating-System) ist für die Arbeitsplatzcomputer (AC) A 7100 und A 7150 des VEB Kombinat Robotron konzipiert. Dieses Betriebssystem unterstützt die effektive Lösung einer Vielzahl wissenschaftlich-technischer und ökonomischer Probleme in allen Teilen der Wirtschaft. Es soll den Nutzern des AC eine Programmierumgebung bieten, die modernsten Ansprüchen genügt. Dies gilt sowohl für die interaktive Nutzerschnittstelle zur Realisierung des Mensch-Maschine-Dialoges mit einem leistungsfähigen Kommandointerpreter, universell nutzbaren Kommunikationsmechanismen und eine modulare Gestaltung aller Dienstprogramme, die eine baukastenartige Verknüpfung der einzelnen Leistungen zuläßt, als auch für die vorhandenen Programmierwerkzeuge. Die Arbeit mit Files (Dateien) ist problemlos. Der Weg von der Programmübersetzung bis zum Programmstart gestaltet sich äußerst einfach. Als höhere Programmiersprache ist zunächst nur die Sprache C implementiert. Durch deren Ausdrucksmittel und Effizienz wird ein Assembler-einsatz auf Nutzerprogrammebene im Prinzip überflüssig. Dem Programmierer steht auch eine umfangreiche Systembibliothek zur Verfügung.
Durch diese Programmierumgebung wird der Anwender weitgehend von Nebensächlichem entlastet, so daß mehr als bei allen bisher existierenden Betriebssystemen das Hauptaugenmerk den zu lösenden Anwenderproblemen gelten kann.
Ein besonderer Vorzug der Programmierumgebung des MUTOS 1700 besteht darin, daß diese Programmierumgebung durch die Systeme PSU und VMX auf ESER-Anlagen, MUTOS 1630 auf K 1630 und andere kompatible Systeme auf anderen Rechnern zur Verfügung steht und auch auf den SKR-Folgesystemen zur Verfügung stehen wird.
Für einen Einsatz in der Prozeßdatenverarbeitung ist MUTOS 1700 nicht geeignet, da keine Echtzeitunterstützung existiert.

G1.2. Einsatzgebiete

Das Betriebssystem MUTOS besteht aus einer Menge von Moduln und Programmen, die folgende Einsatzgebiete unterstützen:

- Programmentwicklung
- Lösung wissenschaftlich-technischer Probleme
- Textverarbeitung
- Lösung kommerzieller Probleme

MUTOS ist leicht erweiterbar und kann gut an bestimmte Einsatzgebiete angepaßt werden. Eine Einbindung neuer Geräte in die Systemkonfiguration ist bei vorliegenden Geräte-Drivern in kürzester Zeit möglich.

G1.3. Bestandteile des Betriebssystems MUTOS

Das Betriebssystem MUTOS umfaßt folgende Bestandteile (Programmgruppen):

- Kern des MUTOS mit folgenden Aufgaben:
  - Nutzerprogrammorganisation
  - E/A-Organisation
  - File-System-Unterstützung

- Dienstprogrammgruppen für folgende Aufgabenkomplexe:
  - File-Manipulation
  - Manipulation von Directories (Dateiverzeichnissen) und File-Namen
  - Abarbeitung von Nutzerprogrammen
  - Nutzerkommunikation
  - Programmentwicklung
  - Arbeit mit implementierten Programmiersprachen
  - automatische Textverarbeitung
  - Bearbeitung großer Datenmengen
  - Steuerung des Nutzerzugriffs und Einstellen von Terminal-Funktionen
  - Statusabfragen
  - Wartung und Betrieb des Systems
  - Leistungsabrechnung

- C-Compiler

- Bibliotheken für die
  - E/A-Unterstützung
  - Programmorganisation
  - Berechnung mathematischer Funktionen

G1.4. Systemphilosophie

MUTOS ist als interaktives Teilnehmersystem konzipiert. Ein wichtiges Ziel dieses Betriebssystems besteht darin, dem Nutzer für die Bearbeitung seiner Aufgaben eine nutzerfreundliche Programmierumgebung anzubieten.
Im Gegensatz zu vielen anderen Betriebssystemen, die eine sehr komplexe und oft unübersichtliche Prozeß- und Speicherverwaltung besitzen oder eine Vielzahl von File-Zugriffsroutinen unterstützen, wurde bei MUTOS in diesen Bereichen Einfachheit und gute Verständlichkeit erreicht. Dies wurde durch die praktisch ausschließliche Verwendung einer höheren Programmiersprache, leicht zu benutzender Systemrufe und auf diesen aufbauender Systembibliotheksroutinen verwirklicht.
Besondere Beachtung verdient in einem interaktiven Teilnehmersystem der Kommandointerpreter. Dort sind gute Verständlichkeit auch für den Anfänger, gepaart mit hoher Leistungsfähigkeit für anspruchsvolle Nutzer gefordert. Auch die Systemprogramme müssen diese Erfordernisse berücksichtigen. Zur Realisierung dieser Systemphilosophie wurde bei MUTOS folgendermaßen verfahren:

- Implementierung eines hierarchischen File-Systems, das um eine beliebige Anzahl von Datenträgern erweitert werden kann.

- Weitestgehende Geräteunabhängigkeit durch syntaktisch gleichartige Behandlung von Files und Geräten. Dieses Konzept wird noch durch die Möglichkeit der Interprozeß-Kommunikation über Pipes ausgedehnt.

- Realisierung einer automatischen Speicherzuordnung und Prozeßauslagerung.

- Implementierung einer effektiven Kommandosprache mit den Ausdrucksmitteln einer höheren Programmiersprache und die Möglichkeit des Austausches des Kommandointerpreters je nach den Bedürfnissen des Nutzers. (An verschiedenen Arbeitsplätzen kann zur gleichen Zeit mit unterschiedlichen Kommandointerpretern gearbeitet werden.)

- Es existiert eine große Anzahl von Dienstprogrammen, die durch universell nutzbare Kommunikationsmechanismen (Systemrufe, Signale, Pipes) des Systems untereinander kombiniert und in die Problemlösung des Anwenders eingebunden werden können.

- Anwendung der höheren Programmiersprache C in der Systemprogrammierung, wo bei bisherigen Betriebssystemen Assemblersprachen benutzt wurden. (Nur etwa 10% des Kerns von MUTOS sind in Assembler, der Rest ist in der Sprache C geschrieben. Dadurch wurde auch eine gute Portabilität des gesamten Betriebssystems erreicht).

2. Gerätekonfiguration

Folgende Mindestkonfiguration benötigt MUTOS 1700 für seine Arbeit:

- AC A 7100 oder AC A 7150 mit
  - . 256 K Byte Hauptspeicher
  - . Controller für Externspeicher (KES)
  - . 2 Minifolienspeicherlaufwerken
  - . Bildschirm
  - . Tastatur

Bedient werden außerdem:

- bis zu 768 K Byte Hauptspeicher
- 2 Folienspeicherlaufwerke im Beistellgefäß (Mini- oder Standardfolienspeicher)
- 1 Anschlußsteuerung für serielles und paralleles Interface (ASP) mit
  - . 1 x IFSS
  - . 1 x S2(V.24)
  - . 1 x IFSP
- 1 Seriendrucker mit Centronics-Interface
- Festplattenspeicher

Die Normalkonfiguration für eine sinnvolle Arbeitsweise sollte aus folgenden Geräten bestehen:

- maximal möglicher Hauptspeicherausbau
- Festplattenspeicher
- mindestens 1 Folienspeicherlaufwerk
- 1 ASP
- 1 Seriendrucker mit Centronics-Interface

Die Externspeicherkapazität sollte so hoch wie möglich gewählt werden. Steht kein Festplattenspeicher zur Verfügung, ist ein Beistellgefäß mit Folienspeicher sehr sinnvoll. Eine Beschränkung auf die beiden Einbaulaufwerke macht die Arbeit mit MUTOS 1700 sehr schwierig. Dabei ist bei Einsatz doppelseitiger Laufwerke immer noch der volle Leistungsumfang nutzbar. Unter ausschließlicher Nutzung zweier einseitiger Minifolienspeicherlaufwerke ist nur eine sehr eingeschränkte Arbeit möglich.
Unabhängig von der Ausrüstung mit Externspeichern wird das Betriebssystem in vollem Umfang auf Disketten zur Verfügung gestellt. Der Anwender kann daraus nach entsprechenden Anleitungen in der Dokumentation und mit den angebotenen Hilfsmitteln das für ihn günstigste System aufbauen.
Die Anschlüsse der ASP können für unterschiedliche Zwecke benutzt werden, so z.B. für weitere Drucker (neben dem Centronics-Anschluß), für Plotter und für Kopplungen zu anderen Rechnern. Auch der Anschluß eines Terminals (z.B. K 8911) für den Mehrnutzerbetrieb ist möglich.

3. Leistungen des Betriebssystemkerns

Der Kern des Betriebssystems MUTOS koordiniert durch eine zentrale Prozeß- und Speicherverwaltung die Abarbeitung aller zu bearbeitenden Prozesse. Damit wird ein effektives Angebot der Systemleistungen gewährleistet. Außerdem wird durch den Kern des MUTOS die Arbeit der einzelnen Nutzer mit dem File-System realisiert. Der dritte große Aufgabenkomplex besteht in der Organisation der Ein- und Ausgaben und in der Behandlung aller Interrupts. Es existieren Geräte-Driver für alle in Abschnitt 2 (Gerätekonfiguration) aufgeführten externen Geräte.
Im folgenden werden das File-System des Betriebssystems, die Ein-/Ausgabe- Organisation und die Prozeßbehandlung etwas näher vorgestellt.

G3.1. File-System des MUTOS

Das Betriebssystem MUTOS unterstützt ein transparentes, leicht zu verwaltendes und hierarchisch strukturiertes File-System. Es bildet einen aus Directories und Files bestehenden Baum. Die Wurzel dieses Baumes ist die Root-Directory. Jeder Eintrag in einer Directory enthält einen File-Namen, der entweder ein reguläres File, ein Spezial-File oder wiederum eine Directory benennt. Der Platz eines Files im File-System wird durch seinen Pfadnamen eindeutig beschrieben. Er gibt an, wie ein File ausgehend von der Wurzel erreicht werden kann.
Der MUTOS-Kern unterstützt außer den genannten drei File-Arten keine weitere File-Strukturierung. Alle anderen Interpretationen sind programmabhängig.

G3.1.1. Reguläre Files

Reguläre Files enthalten vom Nutzer abgelegte Informationen. Sie werden lediglich als eine Folge von Bytes betrachtet, die einzeln adressierbar sind. Für das Betriebssystem hat der Inhalt der Files keine bestimmte Struktur. Sie wird nur von den verarbeitenden Programmen festgelegt.

G3.1.2. Directories

Directories sind Verzeichnisse von File-Namen und zugehörigen i-Nummern (i-numbers). Eine i-Nummer ist nichts anderes als ein Verweis auf einen das File beschreibenden Informationsblock, die sogenannte Inode. Damit ist die Verbindung zwischen File-Namen und den Files selbst hergestellt.
Im allgemeinen hat jeder Nutzer eine eigene Directory. Dort kann er auch Subdirectories anlegen, die Gruppen von Files unter geeigneten Gesichtspunkten zusammenfassen.
Directories werden vom System wie reguläre Files behandelt, allerdings mit der Ausnahme, daß sie nur durch privilegierte Programme beschrieben werden können. Auf diese Weise kann das Betriebssystem ihren Inhalt kontrollieren.
MUTOS verwaltet eine große Anzahl von Directories für den eigenen Gebrauch. So gibt es zum Beispiel die Root-Directory, die die Wurzel des baumstrukturierten File-Systems darstellt. Von der Root-Directory aus ist jedes File durch schrittweises Bewegen durch die File-Hierarchie zu erreichen. Andere System-Directories enthalten zum Beispiel alle Dienstprogramme oder Files, die der

Systemverwaltung dienen. Derartige Vereinbarungen sind modifizierbar, sollten aber nach einmaliger Festlegung beibehalten werden.
File-Namen bestehen aus maximal 14 beliebigen KOI7-Zeichen. Wenn das System einen Zugriff zu einem bestimmten File realisieren soll, muß der Pfadname des Files angegeben werden. Dieser Pfadname ist eine Folge von Directory-Namen, die durch "/" getrennt sind. Er endet mit dem entsprechenden File-Namen. Wenn am Anfang ein "/" steht, beginnt der Weg zum File bei der Root-Directory. So sucht zum Beispiel das System beim Pfadnamen /alpha/beta/gamma zunächst in der Root-Directory die Directory alpha, dort die Directory beta und schließlich in dieser das File gamm_a. Dieses File kann ein reguläres File, wieder eine Directory oder ein Spezial-File sein. Mit dem ersten "/" wird die Root-Directory selbst angesprochen.
Wenn der angegebene Pfadname nicht mit "/" anfängt, beginnt das System die Suche in der aktuellen Directory des Nutzers. Zum Beispiel bezeichnet alpha/beta dann den File-Namen beta in der Subdirectory alpha der aktuellen Directory.
Ein File, das keine Directory ist, kann in verschiedenen Directories unter verschiedenen Namen registriert sein, obwohl es physisch nur einmal existiert. Diese Möglichkeit wird Linking genannt, entsprechend wird eine Directory-Eintragung für Files auch als Link bezeichnet. Alle Links zu einem File haben dabei den gleichen Status. Sie enthalten nur den File-Namen und einen Verweis zur entsprechenden (derselben) Inode. Ein File existiert also im Prinzip unabhängig von einer Eintragung in einer Directory. Wenn allerdings keine Eintragung mehr vorhanden ist, wird sein Speicherbereich freigegeben.
Jede Directory hat von Beginn an zwei Eintragungen, die Namen "." und "..". Der Name "." verweist auf die Directory selbst. Ein Programm kann so die aktuelle Directory unter der Bezeichnung "." lesen, ohne den vollständigen Pfadnamen zu kennen. Der Name ".." verweist auf die Directory, in welcher sie selbst definiert ist.
Die Directory-Struktur ist eine Baumstruktur. Abgesehen von den speziellen Eintragungen "." und ".." gilt, daß für jede Directory genau eine Eintragung in einer anderen Directory existieren muß.

G3.1.3. Spezial-Files

Spezial-Files sind ein Bestandteil des File-Systems von MUTOS, der besonders hervorgehoben werden muß. Zu jedem E/A-Gerät, das vom Betriebssystem unterstützt wird, gehört ein solches Spezial-File. Sie können wie reguläre Files beschrieben oder gelesen werden. Dies bewirkt dann ein Lesen von dem oder ein Schreiben auf das ihnen zugeordnete Gerät.
Jedes Spezial-File hat eine Eintragung in der Directory /dev. Wie bei regulären Files können auch bei diesen in anderen Directories Links zu diesem File existieren. Es gibt Spezial-Files für jede Platte, jede Diskette, jeden Drucker und auch für den Hauptspeicher (wobei natürlich 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 einem Drucker realisiert werden. (Lesen ist hier natürlich unsinnig und führt zu einer Fehlermeldung, da ein entsprechendes Zugriffsrecht nicht existiert.)
Aufgrund dieser Einordnung der E/A-Geräte gilt für File- und Gerätenamen die gleiche Syntax und Semantik. Einem Programm, welches (als Parameter) einen File-Namen erwartet, kann somit an dieser Stelle auch ein Gerätename übergeben werden. Schließlich wird auch für die Spezial-Files derselbe Schutzmechanismus wie für andere Files benutzt (siehe 3.1.5.).

G3.1.4. Erweiterbarkeit des File-Systems

In MUTOS gibt es die Möglichkeit, ein File-System auf einem Datenträger durch ein weiteres File-System auf einem anderen Datenträger so zu erweitern, daß nach außen hin die Erweiterung nicht mehr erkennbar ist. Zum Eingliedern eines File-Systems werden der mount-Systemruf und ein darauf aufbauendes mount-Kommando angeboten. Der Systemruf hat die Form:

```
mount(para1,para2)
```

Dabei ist:

```
para1:  der Name eines Spezial-Files, dem ein Datenträger
        mit einem eigenen File-System zugeordnet ist;
para2:  der Name einer existierenden Directory im
        zu erweiternden File-System.
```

Der mount-Systemruf bewirkt, daß die Eintragung für die angegebene Directory nicht mehr zum ursprünglichen Directory-Inhalt verweist, sondern zum Inhalt der Root-Directory des eingegliederten File-Systems. Das heißt, ein bestimmter Teil des File-Systems wird durch ein auf einem anderen Datenträger befindliches File-System ersetzt.
Bei jedem Neustart des Betriebssystems MUTOS ist eine erforderliche Anzahl File-Systeme, z.B. mit Dienstprogrammen, Bibliotheken oder auch den Nutzer-Files einzugliedern. Dabei wird bei größeren Konfigurationen, z.B. bei einem fest-plattenresidenten System, ein Teil dieser Arbeit bereits automatisch ausgeführt.

G3.1.5. File-Schutzmechanismen

Die Prinzipien des File-Schutzes sind bei MUTOS sehr einfach realisiert, ohne dabei aber an Wirksamkeit zu verlieren.
Jeder Nutzer des Betriebssystems hat eine eigene Nutzeridentifikation (Nutzer-ID), die er vom Systemverwalter für die gesamte Zeit der Arbeit am System zugeordnet erhält. Die Gruppenidentifikation (Gruppen-ID) kann im Zusammenhang mit der kollektiven Bearbeitung größerer Projekte an mehrere Nutzer vergeben werden. Legt der Nutzer ein File an, so wird dieses intern mit der Nutzer- und Gruppen-ID versehen. Darüber hinaus werden für jedes File 12 Schutzbits angelegt, von denen neun dazu dienen, die Zugriffsrechte Lesen, Schreiben und Ausführen für den File-Eigentümer, die anderen Mitglieder seiner Nutzergruppe sowie für die übrigen Nutzer des Systems festzulegen.
Jedes Programm wird mit der Nutzer- und Gruppen-ID des aktuellen Nutzers (aktuelle Nutzer- und Gruppen-ID) abgearbeitet. Bei File-Zugriffen werden diese aktuellen Nutzer- und Gruppen-ID mit den Nutzer- und Gruppen-ID des File-Eigentümers verglichen und daraus die Zugriffsrechte abgeleitet (entsprechend Einordnung des aktuellen Nutzers in File-Eigentümer, Gruppenmitglied oder übriger Nutzer).
Das Setzen der Schutzbits 11 und 12 eines ausführbaren Files führt dazu, daß das Betriebssystem während der Ausführung (und nur während der Ausführung) dieses Files die effektiven Nutzer- und Gruppen-ID des Aufrufes gegen die Nutzer-und Gruppen-ID des File-Eigentümers austauscht. Der Zweck dieses Mechanismus besteht darin, Nutzern den Zugriff zu bestimmten Files nur über bestimmte (meist privilegierte) Programme zu gestatten. Zum Beispiel kann ein solches privilegiertes Programm ein Abrechnungs-File verwalten, welches nur von diesem Programm selbst (und auf keinem anderen Wege) geändert werden soll. Der Nutzer wird gezwungen, dieses Programm zu benutzen, anstatt direkt auf das Abrechnungs-File zuzugreifen.

Dieser Mechanismus wird auch benötigt, um alle Nutzerprogramme ausführen zu können, die bestimmte privilegierte Systemrufe benutzen.
Das Bit 10 (sticky bit) bewirkt, daß das Programm nach seinem Aufruf im Swap-Bereich des Systems verbleibt. Bei häufig benutzten Kommandos führt das zu Effektivitätssteigerungen.
Jeder Nutzer kann bei seinen eigenen Files mit Hilfe von Kommandos sowohl die Zugriffsrechte als auch Nutzer- und Gruppen-ID ändern und entsprechenden Erfordernissen anpassen. Der Systemverwalter hat jeweils alle Rechte aller Nutzer.

G3.1.6. Implementierung des File-Systems

Wie im Abschnitt 3.1.2. bereits erwähnt, enthält eine Directory-Eintragung nur den Namen des Files und eine ganzzahlige i-Nummer (i-number). Soll nun auf ein File zugegriffen werden, wird diese i-number als Index zu einer Eintragung (i-node) in einer Systemtabelle (i-list) benötigt. In der Inode eines Files befindet sich die Beschreibung des betreffenden Files. Sie umfaßt folgende Informationen:

- Nutzer- und Gruppen-ID des File-Eigentümers
- die Schutzbits
- die physischen Adressen der zum File gehörenden Blöcke
- die File-Länge
- Datum der Erstellung, der letzten Benutzung und der letzten Modifikation des Files
- die Anzahl der Links zum File
- eine Markierung, die angibt, ob das File eine Directory, ein reguläres File oder ein Spezial-File ist

Ist ein File geöffnet, sind seine Gerätenummer (gibt das File-System an), seine i-number und ein read/write-Pointer in einer hauptspeicherresidenten Systemtabelle gespeichert. Die Nummer dieser Eintragung wird im System als File-Descriptor geführt. Damit ist eine unmittelbare Verbindung zu den zum geöffneten File gehörenden Informationen gegeben.
Beim Anlegen eines neuen Files wird eine Inode für dieses File erzeugt. Außerdem erhält die entsprechende Directory eine Eintragung, die aus File-Namen und i-number besteht. Wird ein Link zu einem bereits existierenden File erzeugt, wird diese i-number weiter verwendet und der Link-Zähler in der Inode um 1 erhöht. Beim Löschen eines Files wird die entsprechende Directory-Eintragung beseitigt und der Link-Zähler um 1 verringert. Wenn dieser dabei 0 wird (es existiert dann keine Directory-Eintragung mehr mit dieser i-number), werden die zum File gehörenden Blöcke auf dem Speichermedium und der Inode freigegeben.
Die externen Speichermedien, die File-Systeme enthalten, sind in 512 Byte große Blöcke unterteilt. Diese werden logisch von 0 bis zu einer geräteabhängigen oberen Grenze adressiert. In der Inode eines jeden Files ist Platz für 13 Blockadressen. Die 10 ersten dieser Blockadressen verweisen zu den 10 ersten Blöcken des Files. Wenn das File mehr als 10 Blöcke umfaßt, so verweist die nächste Blockadresse auf einen Block, der die Adressen von weiteren, maximal 128 Blöcken enthält (einfach-indirekter Zugriff). Für noch größere Files werden die beiden nächsten Blockadressen (die 12. und 13. Blockadresse) für eine 2- bzw. 3-fache indirekte Adressierung verwendet. Kleine Files werden also im Durchschnitt geringere Zugriffszeiten haben als größere (diese Bevorzugung von kleinen Elementen zieht sich durch das gesamte System, z.B. tritt sie auch in der Prozeßverwaltung auf).
Bei Spezial-Files enthält die erste Blockadresse ein Zahlenpaar, die sogenannten Major- und Minor-Nummern des Gerätes. Die Major-Nummer verweist auf einen Tabelleneintrag, der Informationen über den zuständigen Driver enthält. Die Minor-Nummer wird vom Driver selbst ausgewertet. Das kann z.B. die

Laufwerksnummer eines Diskettenspeichergerätes sein. Es gibt jeweils eine Tabelle für blockorientierte und eine Tabelle für zeichenorientierte Geräte.

G3.2. Ein-/Ausgabe-Organisation

Im Betriebssystem MUTOS erfolgt die Ein- und Ausgabe von und zu den physischen Geräten in der Regel synchron, d.h. aus der Sicht des ausführenden Programms erhält dieses erst dann die Steuerung aus der E/A-Behandlung zurück, wenn alle Ein- oder Ausgaben abgeschlossen sind. Die Datenübertragungen von und zu blockorientierten Geräten werden jedoch - wenn nicht anders angegeben - intern gepuffert. Das bedeutet, daß nicht jede E/A-Operation auch wirklich zu einem physischen Ansprechen des Gerätes führen muß. Die Anzahl von zeitaufwendigen Ein- und Ausgaben von Daten wird somit reduziert. Soll z.B. ein einzelnes Byte in ein File geschrieben werden, wird zunächst überprüft, ob sich der Inhalt des entsprechenden Blockes auf dem Externspeicher bereits im Hauptspeicher befindet. Ist dies nicht der Fall, wird dieser Block eingelesen und das angesprochene Byte verändert. Anschließend erfolgt sofort die Rückkehr zum aufrufenden Programm, obwohl der Datenblock sich noch im Hauptspeicher befindet. Befindet sich der Inhalt des betreffenden Blockes bereits in einem der Systempuffer, so kann das Byte sofort geschrieben werden.
Dieses Prinzip der Pufferung bei der Ein- und Ausgabe erhöht einerseits den Systemdurchsatz, bedingt aber andererseits, daß der logische Zustand eines Files nicht immer mit dem physischen Zustand der Daten auf dem Externspeicher identisch ist. Das kann bei Systemzusammenbrüchen zu Komplikationen führen. Das Betriebssystem hilft sich dadurch, daß in regelmäßigen generierbaren Zeitabständen eine Aktualisierung des physischen Zustandes der Files vorgenommen wird. Es werden aber auch Programmwerkzeuge zur Überprüfung der File-Systeme angeboten.
Neben dieser blockorientierten Datenübertragung gibt es im MUTOS die zeichenorientierte Datenübertragung. Diese erfolgt im Wesentlichen ohne Zwischenspeicherung. Bei einer Reihe von peripheren Geräten ist nur diese Übertragungsart möglich (Drucker, Terminal). Sie werden als zeichenorientierte Geräte gekennzeichnet. Es können aber auch die blockorientierten Geräte wie zeichenorientierte behandelt und damit die Zwischenpufferung ausgeschaltet werden. Derartige Geräte werden als "Raw-Geräte" bezeichnet. Dazu ist aus Anwendersicht nichts weiter notwendig, als ein zusätzliches "r" in den Gerätenamen einzubringen (z.B. "rf" statt "f"). Die Datenübertragung muß allerdings in Einheiten eines Vielfachen der Geräteblockung erfolgen. Sie wird vor allem bei sehr großen Datenmengen angewendet.

G3.2.1. Umlenkung der Ein-/Ausgabe

Das Ziel der Ein-/Ausgaben eines Programmes im Betriebssystem MUTOS sind grundsätzlich das Standard-Eingabe- bzw. das Standard-Ausgabe-File. Fehlermeldungen werden in das Standard-Fehler-File geschrieben. Wenn nichts besonderes vereinbart ist, sind diese Standard-Files mit der Tastatur bzw. dem Bildschirm gekoppelt. Diese Ein- und Ausgaben können jedoch auf andere Geräte oder Files umgelenkt werden.
Die Angaben zur Umlenkung werden nicht an das ausführende Programm weitergereicht, sondern vom Kommandointerpreter - der Shell - durchgeführt und damit für das ausführende Programm selbst nicht sichtbar.

G3.2.2. Fließbandverarbeitung

Das Prinzip der E/A-Umlenkung kann ausgenutzt werden, um Pipelines aufzubauen, d.h. die zu verarbeitenden Daten wie auf einem Fließband zu den verschiedenen Verarbeitungszentren zu transportieren. Dazu ist die Ausgabe eines Programms mit der Eingabe eines anderen Programms zu koppeln.
Nachdem das erste Programm einen Puffer - die Pipe - gefüllt hat, leert ein zweites Programm den Puffer zwecks Weiterverarbeitung der Daten. Das erste Programm kann dann den Puffer erneut füllen. Der Anwender braucht sich jedoch um diese Synchronisation nicht zu kümmern. Das erledigt für ihn das Betriebssystem.

G3.2.3. Die Standard-E/A -Bibliothek

Zur Gewährleistung der Portabilität von in der Sprache C geschriebenen Programmen war es notwendig, eine einheitliche und portable Ein- und Ausgabe-Schnittstelle zur Verfügung zu stellen. Die Funktionen dieser Schnittstelle sind in der Standard-E/A-Bibliothek enthalten.
Der Programmierer verwendet Systemrufe, um diese Dienstleistungen in Anspruch zu nehmen.
Die gesamte Bibliothek ist sehr umfangreich. Zur Erläuterung der E/A-Organisation werden deshalb nur die wichtigsten Systemrufe vorgestellt.
Quelle und Ziel von Datenübertragungen sind Files, die durch File-Descriptoren eindeutig beschrieben werden. Im Standardfall sind das die Descriptoren für die Files Standard-Eingabe (stdin) und Standard-Ausgabe (stdout). Alle anderen Files müssen erst mit einem File-Descriptor verbunden werden, d.h. die Files müssen geöffnet werden.

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

Dabei ist name der Name des zu öffnenden Files. Hier kann ein beliebiger Pfadname angegeben werden. Der Parameter flag gibt an, ob das File zum Lesen, Schreiben oder ändern (Lesen und Schreiben gleichzeitig) geöffnet werden soll. Der Rückkehrwert fildes ist der Descriptor. Das ist eine ganze Zahl, die zur Identifikation des Files in den folgenden Rufen (read, write oder anderen Rufen für die File-Manipulation) benutzt wird.
Um ein neues File anzulegen oder ein altes vollständig zu überschreiben, steht der Systemruf

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

zur Verfügung. Durch die angegebene Anweisung wird das File name erstellt und geöffnet. Zum Schluß wird unter fildes der File-Descriptor abgespeichert.
Zunächst wird nur das sequentielle Schreiben und Lesen von Files betrachtet. Das bedeutet, daß beim nächsten E/A-Ruf nach dem zuletzt geschriebenen (bzw. gelesenen) Byte des betreffenden Files fortgesetzt wird. Dafür existiert ein Pointer, der vom System verwaltet wird und auf das nächste zu lesende bzw. zu schreibende Byte des Files zeigt. Ein geöffnetes File kann nun mit folgenden Systemrufen angesprochen werden:

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

Durch diese Systemrufe werden maximal count Bytes zwischen dem mit fildes spezifizierten File und dem mit buffer angegebenen Puffer übertragen. Die tatsächliche Anzahl transportierter Bytes wird in _n zurückgegeben. Im Normalfall ist n gleich count, bei E/A-Fehlern sind Abweichungen möglich.

Bei einem read-Systemruf ist bei fehlerfreier Ausführung n stets kleiner gleich count. Wird beim Lesen versucht, das File-Ende zu überschreiten, bricht das System die Übertragung an dieser Stelle ab.
Beim Schreiben werden nur die Teile eines Files verändert, die durch den Stand des writewritewrite-Pointers und die angegebene Zeichenanzahl betroffen sind. Der Rest des
Files bleibt unverändert. Wird dabei das File-Ende überschritten, so wird das File um die erforderliche Byte-Anzahl erweitert.
Der wahlfreie Zugriff wird mit Hilfe des read/writee-Pointers realisiert. Dieser wird mit dem Systemruf

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

so verändert, daß als nächstes auf das gewünschte Byte des entsprechenden Files zugegriffen werden kann. Der Pointer, der zum File-Descriptor fildes gehört, wird auf eine Position eingestellt, die durch offset und base definiert ist. Dabei gibt base an, ob als Bezugspunkt der File-Anfang, der aktuelle Pointer-stand oder das File-Ende gilt, während offse_t die Verschiebung bezüglich dieser Basis definiert. Der aktuelle Stand des read/write-Pointers wird in location zurückgegeben.

Das Betriebssystem bietet noch eine Vielzahl von Diensten für die E/A-Arbeit und Manipulation mit dem File-System an, wie z.B. Rufe zum Schließen eines Files, Feststellen des File-Status, ändern der Zugriffsrechte und des Eigentümers, Erstellen einer Directory, Erzeugen eines Link zu einem File und Löschen eines Files.

G3.3. Arbeit mit Prozessen

Ein Prozeß ist im MUTOS nichts anderes als die gesamte Umgebung zur Ausführung eines oder (nacheinander) mehrerer Programme. Die Prozeßverwaltung steuert die Zuordnung der Prozesse zu den einzelnen Systemressourcen. Dabei wird für die ZVE nach dem Zeitscheibenprinzip verfahren.
Dem Anwender werden Systemrufe angeboten, mit denen Prozesse geschaffen oder vernichtet bzw. einzelne Prozeßbestandteile verändert oder ergänzt werden kön-nen (z.B. Ausführen eines anderen Programms, öffnen oder Schließen von Files). Außerdem gibt es Möglichkeiten zur Prozeßsynchronisation.
Bei der Arbeit im Dialog wird beim Aufruf eines Dienstprogrammes vom Komman-dointerpreter (Shell) ein Systemruf abgearbeitet, der einen neuen Prozeß eröff-net. Anschließend wird dieser neue Prozeß mit der Ausführung des aktivierten Dienstprogrammes beauftragt. Diese Möglichkeit der Prozeßerstellung und Pro-grammzuordnung kann aber nicht nur vom Kommandointerpreter, sondern auch von allen anderen System- und Nutzerprogrammen genutzt werden. Im folgenden werden einige Systemrufe zur Arbeit mit Prozessen kurz erläutert.

G3.3.1. Erzeugen eines neuen Prozesses

H
Die einzige Möglichkeit, einen Nutzerprozeß neu zu schaffen, besteht in der Anwendung des Systemrufes

```
processid = fork()
```

Durch diesen Ruf wird der ausführende Prozeß in zwei voneinander unabhängige Prozesse aufgespaltet, die beide zunächst dasselbe Programm (aber voneinander unabhängig) ausführen und eine gleiche Prozeßumgebung haben (z.B. offene Files). Der neue Prozeß (Child-Prozeß) unterscheidet sich vom Originalprozeß (Parent-

Prozeß) nur durch den Rückkehrwert (processid). Während der Systemruf fork im Parent-Prozeß den Prozeßidentifikator des neu erzeugten Child-Prozesses zurückgibt, ist der Rückkehrwert im Child-Prozeß gleich Null.

G3.3.2. Ausführen eines Programms

Ein weiterer wichtiger Systemruf ist

```
execute(file,arg1,...,argn)
```

Dieser Ruf veranlaßt den ausführenden Prozeß, den Inhalt des im Argument f_ile angegebenen Files als Programm auszuführen, wobei arg1 ,..., argn Argumente des auszuführenden Programms sind.
Die Voraussetzung ist, daß f_ile auf ein File verweist, welches ein übersetztes und ausführbares Programm enthält. Bei fehlerhafter Ausführung des execute-Rufes wird ein entsprechender Rückkehrcode zurückgegeben.
Bei einem fehlerfrei abgearbeiteten execute-Ruf gilt das bisher von diesem Prozeß abgearbeitete Programm als beendet und kann nicht mehr angesprochen werden. Der Prozeß wird nun den Inhalt von f_ile als Programm abarbeiten.
Im MUTOS ist es üblich, zunächst mit dem Systemruf fork einen neuen Prozeß zu erzeugen. Sowohl der Parent- als auch der Child-Prozeß führen nun dasselbe Programm (voneinander unabhängig) aus. Im Child-Prozeß (processid=0) kann dann mit dem Systemruf execute die Ausführung eines beliebigen anderen Programms veranlasst werden. Diese Verfahrensweise hat den Vorteil, daß beide Prozesse zunächst über die gleiche Umgebung verfügen und dort auch Manipulationen (z.B. zur Interprozeßkommunikation) vornehmen können.

G3.3.3. Prozeßsynchronisation

Zur Prozeßsynchronisation existiert der folgende Systemruf:

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

Der Prozeß, der diesen Ruf ausführt, wird in einen Wartezustand versetzt, bis einer seiner Child-Prozesse die Arbeit beendet hat. Der Rückkehrwert des Systemrufes wait ist der Prozeßidentifikator dieses Child-Prozesses. Existiert kein Child-Prozeß, so wird ein Fehler angezeigt. Im Parameter status werden genauere Informationen über den Zustand des Child-Prozesses bei seiner Beendigung übergeben.

G3.3.4. Beenden eines Prozesses

Der Systemruf

```
exit(status)
```

beendet die Existenz eines Prozesses und vernichtet alle im Zusammenhang damit existierenden Informationen. Befindet sich der dazugehörige Parent-Prozeß im Wartezustand (wait-Systemruf), so wird dieser über die Beendigung der Existenz seines Child-Prozesses informiert. Dabei wird der Wert von status an den wait-Ruf des Parent-Prozesses übergeben.

4. Kommandosprache Shell

G4.1. Einführung

Die Kommunikation der Nutzer mit dem Betriebssystem MUTOS erfolgt über die Kommandosprache Shell. Der diese Sprache verarbeitende Kommandointerpreter hat keinen besonderen Vorrang innerhalb der Bestandteile von MUTOS. Er ist ein ausführbares Programm, das eine Nutzerschnittstelle zur interaktiven Arbeit realisiert und auslagerbar ist wie alle anderen Dienstprogramme.
Der volle Umfang der Kommandosprache ist relativ groß. Die Shell bietet die vielfältigsten Möglichkeiten, um dem Nutzer die Arbeit unter dem Betriebssystem MUTOS zu erleichtern.
Die Shell unterstützt u.a.:

- verschiedene Elemente, wie sie aus algorithmischen Sprachen bekannt sind:
  . Anweisungen zur Steuerung des Programmablaufs (for, case, if, while)
  . Variable
  . Parameterübergaben
- Kommando- und Parametersubstitution
- File-Namen-Generierung
- Umlagerung der E/A-Richtungen
- Modifikation der Umgebung, in der ein Prozeß abläuft
- Signalbehandlung
- Datenübergabe zwischen verschiedenen Prozessen über Pipes
- Suchen von Files entlang der vom Nutzer definierten Pfade innerhalb des File-Systems

Die Shell-Syntax zeichnet sich durch eine knappe Notation aus, die aus leicht verständlichen und einprägsamen Zeichen aufgebaut ist. Unter Ausnutzung der anwenderfreundlichen Eigenschaften von MUTOS, wie der Transparenz des File-Systems, der vielen Dienstprogramme, die vom Standard-Eingabe-File stdin lesen und zum Standard-Ausgabe-File stdout ausgeben, sowie des Pipe-Konzeptes, bietet die Shell dem Nutzer bei wenig Eingabearbeit viel an Systemleistung. Dabei unterstützt der Kommandointerpreter sowohl die interaktive Arbeit, d.h. die Eingabe einzelner Kommandozeilen über das Bildschirmterminal des Nutzers, als auch die Abarbeitung 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 völlig genügt, einige wenige Elemente der Kommandosprache und die Wirkungsweise einiger Dienstprogramme von MUTOS zu kennen und später bei entsprechendem Bedarf nach und nach weitere Möglichkeiten der Shell hinzuzulernen.

G4.2. Shell-Programmierung

G4.2.1. 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 Kommandoname. Im allgemeinen ist dies der Name eines Files, welches ein ausführbares Programm enthält. Dieses Programm wird als Ergebnis der Interpretation des angegebenen Kommandos abgearbeitet. Die verbleibenden Worte aus dem Kommando werden durch die 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 können. So wird nach Eingabe von

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

ein File mit dem Namen comname gesucht und über einen executeexecuteexecute-Systemruf dessen
Ausführung eingeleitet, wobei die Argumente arg1, arg2, ...,argn als Parameter dieses Systemrufes an das auszuführende Programm übergeben werden.

G4.2.2. Hintergrundkommandos

Zur Ausführung eines Kommandos wird von der Shell ein neuer Prozeß eröffnet. In diesem neuen Prozeß erfolgt nun die Abarbeitung dieses Kommandos, während sich der Prozeß, in welchem die Arbeit des Kommandointerpreters abläuft, in einem Wartezustand befindet. Dieses Warten dauert solange, bis die Ausführung des Kommandos beendet ist. Während dieser Zeit ist für den Nutzer keine Interaktion mit der Shell möglich. Wird ein Kommando jedoch mit dem Operator "&" abgeschlossen, ist der Wartemechanismus für dieses Hintergrundkommando außer Kraft gesetzt und die interaktive Arbeit kann unmittelbar nach dem Absetzen des auszuführenden Kommandos fortgeführt werden. Die Eingabe der Kommandozeile

```
cc source.c &
```

bewirkt zum Beispiel, daß der Nutzer während der Übersetzung des C-Programms source.c andere Arbeiten an seinem Bildschirmterminal erledigen kann.

G4.2.3. Umlagerung der E/A-Richtungen

Die meisten Dienstprogramme von MUTOS sind so beschaffen, daß sie Eingaben von einem als Standardeingabe (stdin) bezeichneten File lesen und ihre Ausgaben zu einem als Standardausgabe (stdout) bezeichneten File senden. Für einen Nutzer, der unter Steuerung der Shell arbeitet, werden stdin und stdout zu Beginn der Arbeit seinem Bildschirmterminal zugeordnet (Spezial-File /dev/tty?), so daß die interaktive Arbeit möglich wird. Um Ein-/Ausgaben von/nach einem anderen Gerät als dem Terminal bzw. aus/zu irgendeinem anderen File aus dem File-System von MUTOS zu realisieren, ist mit Hilfe der Umlagerungszeichen "<" und ">" eine änderung der E/A-Richtung für die Dauer der Abarbeitung eines Kommandos vorzunehmen. Zum Beispiel bewirken die Kommandos

```
pr text
```

die Ausgabe des Files text auf das Bildschirmterminal (ohne Umlagerung) und

```
       pr text >/dev/lp
```

die Ausgabe von text auf dem Zeilendrucker (Spezial-File /dev/lp).
Analog dazu wird die Eingaberichtung mit "<" umgelagert.

G4.2.4. Pipelines

Die Bereitstellung des Pipe-Mechanismus durch MUTOS und die einfache Handhabung, die die Shell dazu bietet, ermöglichen auf unkomplizierte Weise die Kombination von Dienstprogrammen des MUTOS untereinander und mit Nutzerprogrammen.
Kommmandos, die durch den Operator "|" verknüpft sind, wie in

```
       anycommand | sort | pr
```

werden als Pipeline bezeichnet. Eine derartige Verknüpfung bewirkt, daß die Kommandos simultan abgearbeitet werden, wobei die Standardausgabe jedes Kommandos als Standardeingabe 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 mit pr formatiert ausgegeben werden.

G4.2.5. Programmablaufsteuerung

Die Kommandos liefern bei Beendigung ihrer Arbeit einen Rückkehrcode. Dieser dient als Testwert zur Programmablaufsteuerung in den Steuerstrukturen

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

und

```
       while Kommandoliste1
       do Kommandoliste2
       done
```

Ihre genaue Arbeitsweise soll hier nicht erläutert werden. Sie ist jedoch analog zu ähnlichen Formen in höheren Programmiersprachen.
Daten- bzw. zeichenkettengesteuerte Verzweigungs- bzw. Schleifenmechanismen werden durch die Steuerstrukturen

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

und

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

bereitgestellt.
Bei der case-Anweisung wird die Zeichenkette Wort nacheinander mit den Mustern Muster1, Muster2, ... verglichen und bei festgestellter Übereinstimmung die zugehörige Kommandoliste abgearbeitet. In der for-Anweisung wird die Shell-Variable Name nacheinander mit den Werten Wort1, Wort2, ... belegt und für jede Belegung die Kommandoliste abgearbeitet.
Mit Hilfe dieser Steuerstrukturen kann auf unterschiedliche Weise der Programmablauf in Shell-Prozeduren beeinflußt werden.

G4.2.6. Shell-Prozeduren

Der Kommandointerpreter Shell kann selbst in einem Kommando aufgerufen werden. Damit besteht die Möglichkeit der Abarbeitung von (nichtinteraktiven) Kommandos, 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 ausgeführt werden. Die Argumente arg1, arg2, ... sind dabei in der angegebenen Reihenfolge den Stellungsparametern $1, $2,... zugeordnet und unter diesen Namen in der Shell-Prozedur verfügbar. Das Kommando sh kann unter Wirkung einer ganzen Reihe von Optionen ausgeführt werden, die z.B. Mechanismen zur Steuerung der Abarbeitung von Shell-Prozeduren bereitstellen.
Falls dem File procname (mit dem Kommando chmod) das Attribut "ausführbar" zugewiesen wurde, kann die Shell-Prozedur ohne Aufruf des Kommandos sh abgearbeitet werden. Die Kommandozeile hat dann einfach die Gestalt

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

wie ein "normaler" Kommandoaufruf.
Shell-Prozeduren sind ein wirksames Mittel zur Vereinfachung der Arbeit am Rechner. Da sie keiner Übersetzung bedürfen, sind sie leicht zu erstellen und zu verwalten. Mit Hilfe besonderer Shell-Optionen wird dem Nutzer das Testen seiner Prozeduren erleichtert.
Die Standardeingabe und die Standardausgabe einer Shell-Prozedur bleiben während der Abarbeitung dieser Prozedur unverändert. Damit können solche Prozeduren in Pipelines benutzt werden.

G4.2.7. Shell-Variable

Sowohl in Prozeduren als auch während der interaktiven Arbeit können Zeichenketten als Werte den Shell-Variablen zugewiesen werden. Auf diese Weise läßt sich z.B. das Schreiben häufig benutzter Namen verkürzen. 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 können mit speziellen Shell-Kommandos an nachfolgend auszuführende Prozeduren "exportiert" werden, wie in

```
export source
```

oder auch für die Dauer einer Prozedur vor Veränderung geschützt werden, wie durch

```
readonly source
```

Zur Vereinfachung der Arbeit mit den Shell-Variablen existieren die Operatoren "-", "=", "?", und "+", die das Testen und Setzen von Variablen verbinden. Außerdem gibt es spezielle Variablen, die von der Shell bereits mit bestimmten Werten belegt sind und vom Nutzer verwendet werden können.

G4.2.8. Kommandosubstitution

Eine Zeichenkette, die in Akzentzeichen (`...`) eingeschlossen ist, wird als Kommando betrachtet, das vor Abarbeitung der gesamten zugehörigen Kommandozeile ausgeführt und durch seine nach der Standardausgabe gerichtete Ausgabe ersetzt werden soll. Das Kommando pwdpwdpwd zum Beispiel liefert als Ausgabe auf die Standardausgabe den Namen der aktuellen Directory. Wenn nun /usr/wrk/th die 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 Möglichkeit, Programme, die Zeichenketten verarbeiten, unkompliziert in Shell-Prozeduren einzubeziehen. Durch die Kommandosprache selbst werden das Zusammenfügen von Zeichenketten sowie das Mustererkennen und -ausfüllen im Zusammenhang mit der File-Namen-Generierung realisiert.

G4.2.9. File-Namen-Generierung

Wird ein File-Name innerhalb eines Kommandos nicht vollständig, sondern in Form eines Musters unter Benutzung der File-Namen-Erweiterungszeichen "*", "?", "[...]" angegeben, so ermittelt die Shell alle dem Muster entsprechenden File-Namen und setzt diese, alphabetisch geordnet, als einzelne Argumente anstelle des Musters in die Kommandozeile ein. Das Zeichen "*" steht dabei für jede beliebige Zeichenkette, "?" für ein beliebiges Zeichen und anstelle von "[...]" kann im zu ermittelnden File-Namen irgendeines der in den eckigen Klammern angegebenen Zeichen auftreten. Daher listet das Kommando

```
ls *.c
```

alle File-Namen in der aktuellen Directory aus, die auf ".c" enden und

```
ls /dev/tty?
```

liefert die Namen aller Terminals (tty1, ..., tty9), für die in /dev eine Eintragung vorhanden ist.

G4.3. Implementierung

Ein vom Nutzer an seinem Bediengerät eingegebenes Kommando bzw. ein aus einer Shell-Prozedur eingelesenes Kommando wird von der Shell zunächst auf seine syntaktische Richtigkeit geprüft. Wenn keine Fehler festgestellt werden, erfolgt die Ausführung aller angegebenen Substitutionen (Parametersubstitution, Kommandosubstitution, File-Namen-Generierung) und die Zerlegung des Kommandos in seine einzelnen Bestandteile. Damit hat das Kommando für seine später erfolgende Abarbeitung mit einem execute-Systemruf die nötige Form erhalten.

An dieser Stelle wird von der Shell mit einem fork-Systemruf ein neuer Prozeß eröffnet. Dieser Child-Prozeß "erbt" von seinem Parent-Prozeß (der arbeitenden Shell) die geöffneten Files, mindestens diejenigen, die als Standard-Eingabe und Standard-Ausgabe benutzt werden, und arbeitet asynchron zu ihm weiter. Außer bei der Abarbeitung im Hintergrund wartet der Parent-Prozeß (Shell) auf die Beendigung des gestarteten Child-Prozesses, d.h. auf die Beendigung der veranlaßten Kommandoabarbeitung. Im Child-Prozeß werden zunächst die ggf. erforderlichen E/A-Umlagerungen durchgeführt. (Im Parent-Prozeß bleibt der alte Zustand bzgl. der E/A-Richtungen erhalten!) Danach wird mit einem execute-Systemruf die Abarbeitung des angegebenen Kommandos gestartet.
Aus einem erfolgreichen execute-Ruf gibt es keine Rückkehr in den diesen Ruf sendenden Prozeß. Der Child-Prozeß wird daher mit Abschluß der Kommandoabarbeitung beendet, und die Arbeit der Shell wird (im Parent-Prozeß) fortgesetzt. Die Shell bekundet durch Anzeige der Eingabeanforderung (Prompt-Zeichen, i.a. "$") auf dem Bildschirm ihre Bereitschaft, ein neues Kommando einzulesen bzw. arbeitet das folgende Kommando in einer Shell-Prozedur ab.
Der Hauptprozeß eines Nutzers, d.h. der Shell-Prozeß, über den der Nutzer zu Beginn seiner Arbeit am Terminal die Kommunikation mit MUTOS aufnimmt, ist selbst Child-Prozeß eines anderen Prozesses. Dieser andere Prozeß wurde als letzter Schritt während der Initialisierung des Betriebssystems gestartet. Er arbeitet das Programm init ab, das für jeden Terminalkanal einen Prozeß erzeugt, wobei die Terminals für Ein- und Ausgaben über die File-Deskriptoren 0, 1 und 2 geöffnet werden. Anschließend wird an jedem Terminal das Programm getty gestartet. Es fixiert die Terminal-Parameter und fordert dann zum Anmelden (login) auf. Die Antwort vom Terminal wird vom Login-Programm ausgewertet.
Nach dem erfolgreichen Eintragen eines jeden Nutzers in das laufende Betriebssystem wird in dessen Home-Directory, eine im Schutzwort-File (/etc/passwd) für jeden Nutzer festgelegte Directory, verzweigt. Die Nutzeridentifikation (Nutzer-ID) des laufenden Prozesses wird gleich der Nutzer-ID des eingetragenen Nutzers gesetzt und die Shell (der Hauptprozeß dieses Nutzers) gestartet.
Die Shell arbeitet zuerst, falls vorhanden, die Shell-Prozedur .profile in der Home-Directory des Nutzers ab. Dort sind Kommandos zusammengefaßt, die jeweils vor Beginn der Arbeit dieses Nutzers ausgeführt werden sollen. Danach ist die Shell bereit, die Eingaben des Nutzers zu verarbeiten.
Dieser initialisierte Haupt-Shell-Prozeß eines Nutzers wartet die meiste Zeit auf Eingaben vom Nutzerterminal. Durch die Eingabe der End-of-File-Zeichenfolge kann dieser Hauptprozeß wieder beendet werden. Dadurch wird init veranlaßt, an dem entsprechenden Terminal die Login-Prozedur zu starten. Der nächste Nutzer kann seine Arbeit beginnen.

5. Systemprogrammgruppen

MUTOS stellt eine Vielzahl von Dienstleistungen über Systemprogramme (Dienstprogramme) zur Verfügung. Diese existieren getrennt vom MUTOS-Systemkern, und demgemäß sind unterschiedliche Ausbaustufen möglich. Entsprechend der MUTOS-Systemphilosophie existieren keine großen Programme mit vielen Dienstleistungen, sondern eine Vielzahl kleinerer Programme mit begrenztem, genau umrissenem Leistungsumfang. Die Flexibilität insbesondere gegenüber Modifizierungen und Ergänzungen ist dadurch sehr hoch. Die Dienstprogramme sind über ein entsprechendes Kommando aufzurufen. Dabei können durch entsprechende Optionen bzw. bei einigen ganz wenigen Dienstprogrammen (z.B. dem Editor) auch durch weitere Kommandos die geforderten Dienste genauer klassifiziert werden.
Die Systemprogramme lassen sich entsprechend den geleisteten Diensten in verschiedene Gruppen einteilen. Sie realisieren folgende Funktionen:

- Steuerung des Nutzerzugriffs
- Einstellung der Terminalfunktionen
- File-Manipulation
- Manipulation von Directories und File-Namen
- Abarbeiten von Nutzerprogrammen
- Statusabfragen
- Wartung und Betrieb des Systems
- Leistungsabrechnung
- Nutzerkommunikation
- Programmentwicklung (Programmentwicklungswerkzeuge)
- operative Benutzung der Systembeschreibung
- Arbeit mit den implementierten Programmiersprachen
- automatische Textverarbeitung
- Bearbeitung von Datenmengen

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

G5.1. Steuerung des Nutzerzugriffs

login
: bewirkt das Anmelden eines Nutzers im Betriebssystem durch Eingabe eines Namens. Das System überprüft, ob ein Nutzer mit diesem Namen existiert und ob seine Directory durch ein Schutzwort (siehe Dienstprogramm passwd) gesichert ist. Nach ordnungsgemäßer Anmeldung erfolgen unmittelbar darauf die automatische Ausgabe von für den speziellen Nutzer relevanten Informationen und das Starten des Kommandointerpreters Shell (Abschnitt 4) oder eines entsprechenden anderen Programms.

passwd
: dient zum ändern bzw. Installieren eines Schutzwortes (password). Jeder Nutzer kann beim Anmelden (siehe login) den Zugang zu seiner eigenen Directory zusätzlich durch ein von ihm selbst gewähltes Schutzwort sichern. Entsprechendes gilt auch für das System selbst.

newgrp
ermöglicht das ändern des Gruppenidentifikators. Diese Möglichkeit dient der Sicherung von Zugriffsrechten, z.B. beim Übergang von einer Projektgruppe in eine andere.

G5.2. Einstellung der Terminalfunktionen

stty
dient der Einstellung von für die jeweilige Aufgabe optimalen Terminalparametern.

tabs
setzt Tabulatoren am Terminal.

G5.3. Manipulation von Files

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

cptree
kann vollständige File-Hierarchien kopieren.

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

cat
dient zum Verketten von beliebigen Files. Das resultierende File wird normalerweise auf dem Standard-Ausgabegerät ausgegeben, wenn nicht durch Spezifikation andere Verfügungen 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.

split
bewirkt das Trennen von Files.

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

lpr
ist ein spezielles Programm für Druckerausgaben im Mehrnutzerbetrieb (Spooling).

tail
dient der Ausgabe von Files ab einer spezifizierbaren Druckposition.

dd
dient zur Übertragung von Files und zur Umsetzung von File-Formaten. Damit wird u.a. ein Datenaustausch mit anderen Rechnersystemen ermöglicht.

sum
dient zur Blockzählung und zur Ermittlung der Prüfsumme von Files. Es ist damit insbesondere für Kontrollzwecke und zur Auffindung von schadhaften Stellen geeignet.

G5.4. Manipulation von Directories und File-Namen

rm
löscht einzelne Files und ganze Directory-Hierarchien.

ln
vergibt Aliasnamen an Files (Erzeugen von Links).

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

chmod
dient dem ändern von Zugriffsrechten zu Files und Directories.

chown
ändert den Eigentümer eines oder mehrerer Files.

chgrp
ändert die Nutzergruppe (z.B. bzgl. eines bestimmten Projektes), zu der ein File gehört.

mkdir, rmdir
bewirken das Erstellen einer neuen bzw. das Löschen einer vorhandenen Directory.

cd
bewirkt das ändern der aktuellen Directory eines Nutzers.

find
bewirkt das Durchsuchen einer beliebigen Directory-Hierarchie nach speziellen Kriterien (z.B. Erstellungsdatum und vieles andere) und das Heraussuchen der den Kriterien entsprechenden Files. Auf jedes gefundene File kann ein spezifiziertes Kommando zur Anwendung gebracht werden (z.B. Archivieren, Löschen usw.).

G5.5. Abarbeiten von Nutzerprogrammen

sh
dient zum Aktivieren des Kommandointerpreters Shell (siehe Abschnitt 4.), der wesentliche Dienste zum Ausführen von Nutzerprogrammen leistet.

test
prüft die Bedingungen von Ausdrücken, wobei der Ausdruck File-Namen, Zeichenketten oder ganze Zahlen enthalten kann.

expr
dient der Berechnung von Zeichenketten zur Ermittlung von Kommandoargumenten.

wait
   bewirkt das Warten auf die Beendigung von asynchron laufenden Prozessen.

echo
   gibt seine Argumente auf dem Standard-Ausgabegerät aus (spezifizierbar sind auch andere Möglichkeiten). Das Dienstprogramm echoechoecho kann z.B. zur Ausgabe von Diagnoseinformationen in Shell-Prozeduren verwendet werden.

sleep
   dient dazu, die Ausführung eines Kommandos um eine spezifizierbare Zeit zu verzögern.

nice
   bewirkt, daß ein Kommando mit erniedrigter bzw. erhöhter (nur für den Systemverwalter möglich) Priorität ausgeführt wird.

nohup
   ermöglicht ein Weiterarbeiten der mit diesem Kommando gestarteten Prozesse, auch wenn der betreffende Nutzer sich abmeldet.

kill
   beendet den spezifizierten Prozeß.

tee
   dient dazu, Daten zwischen verschiedenen Prozessen zu übergeben und in einem bzw. mehreren Files eine Kopie davon abzulegen.

G5.6. Statusabfragen

H
ls
   dient dem Auslisten von File-Namen aus vorgegebenen Directories mit entsprechend spezifizierbaren Zusatzinformationen.

file
   bestimmt durch eine Reihe von Tests den Typ der spezifizierten Files; bei Text-Files wird auch versucht, die (Programmier-) Sprache zu bestimmen.

date
   gibt das aktuelle Datum und die Uhrzeit aus bzw. wird zur Eingabe dieser Daten benutzt.

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

du
   gibt die Anzahl von Blöcken aus, die durch die Files einer vorgegebenen Hierarchie belegt werden.

quot
   gibt die Blockzahl aus, die jeder Nutzer im angegebenen File-System besitzt.

who
   gibt die Systemnamen der zur Zeit im System angemeldeten Nutzer (siehe login) mit der Bezeichnung ihres Terminals und ihrer Anmeldezeit aus.

ps
   dient der Ausgabe von Informationen über momentan laufende Prozesse.

pstat
   dient der Ausgabe von Informationen über den Zustand des gesamten Systems.

iostat
   dient der Ausgabe von statistischen Informationen über die E/A-Aktivitäten des Systems.

tty
   gibt die Bezeichnung des Terminals aus, an dem sich der anfragende Nutzer angemeldet hat.

pwd
   gibt den Namen der für den anfragenden Nutzer aktuellen Directory aus.

G5.7. Wartung und Betrieb des Systems

mount, umount
   dient dem Eingliedern (bzw. Ausgliedern) von Datenträgern mit vollständigem File-System in den (bzw. aus dem) existierenden Directory-Baum.

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

mknod
   dient zum Einrichten einer File-System-Eintragung (i-node) für ein Spezial-File.

tar
   ermöglicht das Verwalten von File-Archiven auf Externspeichern.

dump
   bewirkt, daß von einem File-System ein Abzug auf einem angegebenen Gerät erstellt wird.

dumpdir
   gibt den Inhalt eines mit dumpdumpdump erstellten File-System-Abzugs aus.

touch
   aktualisiert das Datum der letzten Modifikation eines Files.

restor
   stellt ein File-System aus einem Abzug (siehe dumpdumpdump) wieder her. Bei Angabe eines entsprechenden Schlüssels ist dies auch selektiv möglich.

su
   ermöglicht einem Nutzer, zeitweilig den privilegierten Status eines System-verwalters zu erlangen. Dieser ist normalerweise durch ein Schutzwort gesichert (siehe passwd).

dcheck, icheck, ncheck
   dienen zur Kontrolle des File-Systems. Gleichfalls ist eine Wiedergewinnung nicht ordnungsgemäß erfaßter Blöcke (Reparatur des File-Systems) möglich.

fsck
   prüft den Zustand von File-Systemen und stellt bei Ermittlung eines Fehlers den ursprünglichen Zustand vollständig oder teilweise wieder her. Dies geschieht automatisch oder im Dialog.

clri
   entfernt fehlerhaft belegte Blöcke aus dem File-System und ermöglicht im Zusammenwirken mit den checkcheckcheck-Dienstprogrammen Reparaturen des File-Systems.

sync
   veranlaßt, daß alle noch ausstehenden E/A-Aktivitäten des Systems zu Ende gebracht werden und das File-System auf der Magnetplatte vom Hauptspeicher aus aktualisiert wird (Herausschreiben des Superblocks).
   Das Dienstprogramm ist zu benutzen, um bei Beendigung der Arbeit mit MUTOS einen ordnungsgemäßen Systemabschluß zu erzielen.

G5.8. Leistungsberechnung

H
ac
   bewirkt eine Aufstellung über alle Nutzzeiten. Dabei sind mehrere Ausgabemöglichkeiten spezifizierbar.

sa, accton
   Das Dienstprogramm sasasa ermöglicht die Ausgabe der Nutzzeiten des Kommandointerpreters Shell einschließlich einer Information über jedes benutzte Kommando.
   Das Dienstprogramm acctonacctonaccton ermöglicht das Ein- bzw. Ausschalten der Registrierung der für sa erforderlichen Daten.

G5.9. Nutzerkommunikation

mail
   bewirkt, daß eine Nachricht von einem Nutzer an einen oder mehrere andere Nutzer übergeben wird. Das Vorhandensein einer Nachricht wird beim Anmelden (login) angezeigt.

write
   unterstützt die direkte Terminalkommunikation mit einem anderen Nutzer.

wall
   überträgt eine Nachricht an alle angemeldeten Nutzer.

mesg
   unterdrückt den Empfang von Nachrichten, die mit write und wall gesendet wurden.

G5.10. Programmentwicklung

ar
dient zur Verwaltung von Gruppen von Files, die aus Effizienzgründen zu Archiven bzw. Bibliotheken zusammengefaßt sind. Sie werden hauptsächlich vom Lader, aber auch von anderen analogen Aufgaben benutzt.

as
ist der Assembler des Betriebssystems. Er hat im MUTOS im Vergleich zu anderen Systemen eine äußerst untergeordnete Bedeutung, da seine Aufgaben vom C-Programmiersystem wahrgenommen werden (Abschnitt 7.).

adb
ist ein interaktives Testsystem, das u.a. folgende Leistungen unterstützt:
- interaktives Testen mittels Unterbrechungspunkten,
- Analyse von Speicherabzügen nach Programmabbruch (post mortem dump),
- Suche nach vorgegebenen Mustern,
- symbolische Bezugnahmen auf lokale und globale Variable.

basename
entfernt aus einer spezifizierten Zeichenkette alle auf "/" endenden Präfixe und den als Kommandoparameter angegebenen Suffix.

od
ermöglicht die byte- oder wortweise Ausgabe von beliebigen Files in verschiedenen Formaten (Oktal-, Dezimal-, Hexadezimal- und Textformat).

ld
ist der Lader für Objektprogramme. Er bietet die Möglichkeit, verschiedene Objekt-Files zu verbinden und die erforderlichen Routinen aus den spezifizierten Bibliotheken einzufügen.

lorder
dient zur Erstellung einer zum Laden geeigneten Reihenfolge einer Objektprogrammbibliothek, indem gegenseitige Abhängigkeiten berücksichtigt werden.

nm
ermöglicht die Ausgabe der Symboltabelle 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-Files von Assembler und Lader hinzugefügt werden. Ist das Programm ausgetestet, kann somit Platz gespart werden.

time
dient zur Ermittlung der bei der Ausführung eines als Parameter angegebenen Kommandos benötigten Zeiten.

prof
liefert ausführliche statistische Informationen über die Ausführungszeiten von Programmen.

make
dient der Steuerung bei der Erstellung von großen Programmen. Dabei werden ein Steuer-File zur Spezifikation von Quell-File-Abhängigkeiten und die in den Quell-Files vermerkten Daten der letzten änderung benutzt, um daraus den minimalen Aufwand zur Erstellung einer neuen Programmversion abzuleiten.

G5.11. Operative Benutzung der Systembeschreibung

man
ermöglicht die Ausgabe von spezifizierten Teilen des Teils 1 der Programmtechnischen Beschreibung der MUTOS-Dokumentation (Manual) - insbesondere von Beschreibungen der einzelnen Kommandos - als Hilfe während der Arbeit am Terminal.

G5.12. Anwendung der Programmiersprachen

cc
aktiviert nacheinander Precompiler, C-Compiler, Assembler und Linker (sofern keine Fehler erkannt wurden).

cb
ist ein Formatierungsprogramm für C-Programme. Durch systematische Absatzbildung und Klammernanordnung kommt die Struktur eines Programms deutlicher zum Ausdruck.

m4
ist ein universeller Makroprozessor. Er erkennt Makros an beliebiger Stelle im Text. Seine Syntax gestattet eine breite Anwendung.

G5.13. Anwendung der automatischen Textverarbeitung

ed, em
sind interaktive Texteditoren und gleichzeitig universelle Textbearbeitungsprogramme. Sie dienen auch zur Programmaufbereitung. Der Editor emememem bietet gegenüber ededed einen erhöhten Leistungsumfang.

look
sucht nach Worten in einem sortierten File (Wörterbuch), die mit einer spezifizierten Zeichenfolge beginnen.

nroff
ist ein Textformatierungsprogramm. Es ermöglicht ein sehr komfortables Erstellen von Druckdokumenten, wie z.B. der vorliegenden Anwendungsbeschreibung. Die Eingabe besteht aus Textzeilen und Makrozeilen zur Steuerung der Textformatierung.
Es wird ein Standardmakrosatz zur Erstellung von Manuskripten angeboten.

col
ist ein Hilfsprogramm zum Drucken. Es entfernt u.a. nichtdruckbare Zeichen aus Text-Files.

neqn, checkeq
dienen zum Setzen mathematischer Gleichungen und Formeln. Die Formatierungsmakros werden mit einem Preprozessor in detaillierte Satzinstruktionen umgesetzt.

tbl
ist ein Preprozessor für nroff, der einfache Beschreibungen von Tabellen einschließlich ihrer Inhalte in Satzinstruktionen umsetzt.

deroff
bewirkt, daß aus einem vorgegebenen File alle Steuerzeilen für nroff, tbl und neqn entfernt werden.

G5.14. Bearbeitung von Datenmengen

sort
bewirkt zeilenweises Sortieren von Text-Files. Möglich sind lexikographisches und numerisches Sortieren in fallender oder steigender Reihenfolge entsprechend dem spezifizierten Teil der Kommandozeile.

tsort
topologisches Sortieren, d.h. Erstellung einer geordneten Liste von Zeichenkettenpaaren (items).

uniq
dient zum Auffinden und Entfernen von mehrfach auftretenden nacheinander folgenden Zeilen in einem File.

prep
dient zur statistischen Aufbereitung von Texten, indem das spezifizierte File in Wörter eingeteilt und untersucht wird.

rev
kopiert das angegebene File und kehrt die Reihenfolge der Zeichen in jeder Zeile um.

tr
bewirkt die zeichenweise Umcodierung entsprechend einem beliebigen Schlüssel. Darüber hinaus ist auch das Löschen von ausgewählten Zeichen möglich.

diff
stellt alle Unterschiede zwischen zwei vorgegebenen Files fest.

comm
untersucht zwei vorgegebene sortierte Files auf gleichartige Zeilen. In der Ausgabe ist erkennbar, ob eine Zeile jeweils nur in einem der Files oder in beiden enthalten ist.

grep, egrep, fgrep
suchen alle Zeilen eines vorgegebenen Files, die eine spezifizierte Zeichenfolge enthalten.

wc
realisiert das Zählen von Zeilen, Worten (hier: durch Leerzeichen getrennte Zeichenketten) bzw. Zeichen eines Files.

awk
unterstützt das Arbeiten mit der Mustererkennungs- und Musterverarbeitungssprache awk. Alle Zeilen eines Eingabe-Files, die mindestens ein Muster aus einer Menge vorgegebener Muster - auch logische Verknüpfungen sind möglich - enthalten, werden jeweils bestimmten spezifizierten Operationen unterworfen.

6. Fernkopplung

Das Programmpaket uucp bildet eine Netzwerkunterstützung für den Aufbau von Rechnernetzen aus Rechnern mit einem zu MUTOS 1700 kompatiblen Betriebssystem (z.B. K 1630/MUTOS 1630, BC A 5120.16/ MUTOS 8000, SKR - Rechner). Beim Anschluß an solche Rechnernetze fungiert der AC A 7100 oder AC A 7150 als Endrechner, von dem aus eine Kommunikation mit beliebigen Rechnern im Netz möglich ist. Im einfachsten Fall können zwei AC miteinander oder ein AC mit einem der o.g. Rechner gekoppelt werden.

Die Kopplung erfolgt über die seriellen Schnittstellen der ASP K 8071 des A 7100 im Asynchronbetrieb:

- IFSS (Übertragungsentfernung bis ca. 1 km, Übertragungsgeschwindigkeit 2400 oder 4800 Baud)

- V.24, Duplexverbindungen (mit GDN ca. 30 km und maximal 2400 Baud, mit MODEM beliebige Entfernung, max. 2400 Baud)

Durch uucp kann die File-Übertragung zu oder von einem entfernten Rechner zur Kommandoausführung auf diesem und auch die lokale Kommandoausführung unter Beteiligung von Files des entfernten Rechners angewiesen werden. Dadurch ist eine verteilte Datenverarbeitung realisierbar. uucp hat den Status eines Anwendungsprogrammes und arbeitet mit dem Terminal-Driver des MUTOS 1700 zusammen. Zur Anwendung des Programmkomplexes uucp braucht der Nutzer keine speziellen DFV-Kenntnisse zu besitzen, ihm muß lediglich der Systemname des entfernten Rechners bekannt sein.

Im Grundzustand sind die gekoppelten Rechner gleichberechtigt, eine Datenübertragung einzuleiten. Bei Aktivierung des Programmkomplexes uucp (Kommandos uucp oder uux) in einem Rechner führt dieser den Verbindungsaufbau durch. Nach ordnungsgemäßer Datenübertragung bzw. Kommandoausführung erfolgt die Auflösung der Verbindung, falls auch im anderen Rechner keine weiteren Aufträge vorhanden sind. Anderenfalls werden jetzt dessen Aufträge abgearbeitet. Durch den Programmkomplex uucp wird gewährleistet, daß ein Auftrag für einen Partner an einer Verbindungsleitung erst beendet wird, bevor der nächste aktiviert wird.

Zur Sammlung der Arbeitsaufträge bzw. der zu übertragenden Daten verfügt jedes teilnehmende System über eine Spool-Directory. Die Datenübertragung selbst wird im Hintergrund durchgeführt, d.h. der Nutzer des Rechners kann parallel andere Aufgaben bearbeiten. Über die Beendigung der Datenübertragung kann sich der Nutzer über mail oder spezielle Protokoll-Files (LOGFILE bzw. SYSLOG) informieren.

Die Daten bzw. die Datenübertragungsaufträge verbleiben solange in der Spool-Directory des jeweiligen Rechners, bis sie ordnungsgemäß Übertragen oder mit Hilfe des Kommandos uuclean gelöscht worden sind.

Die Übermittlung von Steuermitteilungen und Daten zwischen den gekoppelten Rechnern erfolgt über ein Datenübertragungs- und Sicherungsprotokoll, den Paket-Driver. Das Programmpaket uucp gestattet eine Steuerung der Zugriffsberechtigung für entfernte Rechner oder Nutzer. Auch die von entfernten Rechnern am lokalen System ausführbaren Kommandos können festgelegt werden.

Das Programmpaket uucp besteht aus vier primären und fünf sekundären Programmen. Als primär werden hier diejenigen Programme bezeichnet, die direkt am Datenaustausch bzw. an der entfernten Kommandoausführung beteiligt sind. Die sekundären Programme sind Dienst- und Hilfsprogramme.

G6.1. Primäre Programme

uucp
: erstellt, ausgehend von einer Analyse des gegebenen Kommandos, Arbeits-Files (Daten-Files und Kommando-Files) in der Spool-Directory, die die geforderte Datentransferoperation genauer charakterisieren und später vom Programm uucico zur Ausführung der eigentlichen Übertragung benutzt werden.

uux
: analysiert einen Auftrag zur Kommandoausführung, wenn das ausführende System bzw. eines oder mehrere Files entfernt sind und erstellt in der Spool-Directory entsprechende Files.

uucico
: prüft die Spool-Directory auf vorhandene Eintragungen, bereitet die Übertragung vor und führt dann die Kommando-Files zur Datenübertragung aus. Die Vorbereitung der Übertragung beinhaltet die Ermittlung des zu nutzenden Ausgabekanals, die Errichtung der Verbindung, das Anmelden am entfernten System, den Start des entfernten uucico und die Ermittlung des für beide Systeme geeigneten Übertragungsprotokolls.

uuxqt
: führt die von uux erstellten Files zur entfernten Ausführung von MUTOS-Kommandos aus.

Als Kommando-Interface für den Nutzer werden i.a. nur uucp und uux benutzt. Die Programme uucico und uuxqt werden von uucp bzw. uux automatisch gestartet. Für Testzwecke ist jedoch auch ein getrennter Start von uucico und uuxqt möglich.

G6.2. Sekundäre Programme

uuname
: zeigt die Namen aller im lokalen System bekannten entfernten Rechner an.

uulog
: vereinigt mehrere LOGFILE's, die bei einer parallelen Ausführung von uucp- bzw. uucico-Kommandos entstehen können, in ein LOGFILE und zeigt die Ausführung von uucp-Kommandos betreffenden Statusinformationen (Eintragungen im LOGFILE) an.

uuclean
: löscht in der Spool-Directory enthaltene Files, die älter als eine anzugebende Zeitspanne sind. Der Eigentümer dieses Files kann über das Löschen informiert werden.

uudelock
: löscht LCK-Files, die durch eine abnormale Beendigung von uucico (kill oder Systemcrash) in der Spool-Directory zurückgeblieben sind. LCK-Files sind Lock-Files, die einen parallelen Zugriff mehrerer Prozesse zu der gleichen

Übertragungsleitung verhindern. Dabei wird vorher geprüft, ob die entsprechenden LCK-Files nicht zu noch aktiven Prozessen gehören.

uukill
führt für uucic spezielle Hilfsoperationen in den Phasen des Verbindungsaufbaus und -abbruchs durch.

7. Beschreibung der Programmiersprache C

C ist eine Programmiersprache für allgemeine Anwendungen mit einfacher Ausdrucksweise, modernen Steuer- und Datenstrukturen und einer reichen Auswahl von Operatoren.
Die Sprache C ist weder hochabstrahierend noch umfangreich und wurde auch für kein spezielles Anwendungsgebiet entworfen. Aber das Fehlen von Einschränkungen und die allgemeine Verwendbarkeit machen C bequemer, flexibler und effektiver für viele Aufgaben als andere Programmiersprachen.
Mit ihr wurde der erfolgreiche Versuch unternommen, die Vorzüge höherer Programmiersprachen mit bewährten Prinzipien guter Assembler-Programmierung zu verbinden.
C ist sehr eng mit dem Betriebssystem MUTOS verbunden, da etwa 90% des Systemkerns von MUTOS selbst, alle Kommandos und alle Dienstprogramme in C geschrieben sind.
Die Programmmiersprache C ist jedoch nicht von einem bestimmten Betriebssystem oder von einem bestimmten Rechner abhängig.
C wird zwar oft als Systemprogrammiersprache bezeichnet, da sich die Sprache sehr gut zur Formulierung von Betriebssystemen eignet; umfangreiche Programme für numerische Verfahren, für die Textverarbeitung und für Datenbankanwendungen wurden jedoch ebenfalls in C realisiert.

G7.1. C-Programmiersystem

G7.1.1. Einführung

Der C-Compiler realisiert folgendes Aufgabenspektrum:

- Modifizierung und Interpretation von Makros
- Syntaxprüfungen
- Erzeugen von Objektmoduln
- Verbinden von Objektmoduln zu einer Phase (Linker-Funktion).

Der C-Compiler wird mit dem Kommando cc aufgerufen. Die Syntax des C-Kommandos hat folgende Form:

```
cc  [ option ] ... file ...
```

Die Optionen steuern den Ablauf der Übersetzung; z.B. können die Linkerphase unterdrückt, der Compileroptimierer aufgerufen und der Name des entstehenden abarbeitungsfähigen Programmes vereinbart werden.
Die in der Argumentliste angegebenen Files werden, wenn deren Namenserweiterung (extension) .c ist, als C-Quellprogramme interpretiert, und wenn die Namenserweiterung .s ist, als Assembler-Quellprogramme angesehen. Durch die Übersetzung entstehen Objektprogramme, die pro Quellprogramm in einem File abgelegt werden, deren Namen aus dem der Quelle bestehen, aber als Erweiterung .o bekommen.

G7.1.2. Preprozessor

C läßt einfache Spracherweiterungen mittels eines Makroprozessors zu, auch Preprozessor genannt. Der Preprozessor kann

- vor der Übersetzung Textersatz vornehmen (Makrosubstitution),
- Files in ein Quell-File einfügen und
- Teile einer Quelle von der Übersetzung ausschließen/auswählen (bedingte Übersetzung).

Beginnt eine Quellzeile mit "#" in der Spalte 1, so wird sie von diesem Preprozessor erkannt und interpretiert. Preprozessorzeilen können überall im C-Programm auftreten und behalten ihre Wirkung bis zum Ende des Quellprogramm-Files, sofern nichts anderes vereinbart wird.

7.1.2.1. Textersatz

Textersatz wird durch eine Zeile der folgenden Form verlangt:

```
#define ident string
```

Der Preprozessor ersetzt vor der Übersetzung ident durch string. Der Textersatz kann auch parametrisiert werden:

```
#define ident(arg 1,...,arg n) string
```

Auch hier wird ident durch string ersetzt. In string wird jedoch jeder Parameter durch den entsprechenden Argumenttext aus dem Makroaufruf ersetzt. Die Anzahl der Argumente und Parameter muß übereinstimmen. Das Löschen einer Makroanweisung ist durch

```
#undef ident
```

möglich.

7.1.2.2. Bedingte Übersetzung

Mit der Preprozessoranweisung

```
#ifdef ident
```

kann man Zeilen im Quell-File ignorieren oder übersetzen lassen. Es wird überprüft, ob momentan ident mit Hilfe einer #define-Anweisung definiert ist.

```
#ifndef ident
```

überprüft, ob ident im Augenblick nicht definiert ist, und

```
#if constant
```

untersucht, ob ein konstanter Ausdruck einen von Null verschiedenen Wert (Bedingung ist wahr) liefert. Nach jeder dieser Bedingungen können beliebig viele Quellzeilen folgen. Darunter kann sich auch jeweils eine Zeile der Form

```
#else
```

befinden. Am Schluß der Konstruktion steht immer folgende Preprozessorzeile:

```
#endif  .
```

Trifft die Bedingung in der Konstruktion zu, dann werden die Quellzeilen zwischen #else, wenn vorhanden, und #endif ignoriert. Trifft die Bedingung in der Konstruktion nicht zu, dann werden die Quellzeilen zwischen der Bedingung und einer #else-Anweisung, wenn vorhanden, oder einer #endif-Anweisung ignoriert. Die Konstruktion kann verschachtelt auftreten. Ein Makro kann auf der Ebene des Compiler-Aufrufes mittels der Option -D angegeben werden.

7.1.2.3. Einfügen von Files

Vom C-Compiler wird die Arbeit mit Quelltexteinschließungen unterstützt. Das Einfügen von Files wird durch den Preprozessor beim Vorhandensein einer Quellzeile der Form

```
#include "name"
```

vorgenommen. Die Suche nach diesem File name beginnt im Arbeitskatalog und wird in Standardkatalogen fortgesetzt. Soll nicht im Arbeitskatalog gesucht werden, sondern nur in Standardkatalogen, wird folgende Form der Anweisung verwendet:

```
#include <name>
```

Ein Katalog kann auch beim Aufruf des Compilers mittels der Option -I eingeführt werden.

7.1.2.4. Zeilensteuerung

Die Zeilensteuerung dient der Unterstützung anderer Preprozessoren bzw. Generatoren, die C-Programme erzeugen (z.B. yacc). Eine Zeile der Form

```
#line constant ident
```

veranlaßt den Compiler anzunehmen, daß für Zwecke der Fehlerdiagnostik durch die Konstante constant die Zeilennummer der nächsten Quellzeile gegeben und durch den Bezeichner ident das aktuelle Eingabe-File für etwaige Fehlermeldungen benannt sind. Fehlt der Bezeichner, so wird der bisher zu diesem Zweck benutzte Name wieder verwendet.

G7.1.3. Getrennte Übersetzung

Ein gut strukturiertes C-Programm besteht neben normalen Anweisungen aus einer Vielzahl von Funktionen (Unterprogrammen). Beim ändern nur weniger Quellprogramme (bzw. eines) brauchen nur jeweils für diese (dieses) ein neues Objekt-File erstellt werden, d.h. die Linker-Phase wird unterdrückt. Auf der Ebene des Compiler-Aufrufes ist dafür die Option -c zu verwenden.

G7.1.4. Der Linker

Der Linker verbindet mehrere Objekt-Files einschließlich notwendiger Funktionen aus Objektbibliotheken zu einem ausführbaren Programm.
Der C-Compiler-Aufruf realisiert diese Aufgaben des Linkers mit, indem beim Compiler-Kommando die Option -o _n_ame angegeben wird, wobei name der Name des ausführbaren Programmes ist. Ist die Option -o nicht angegeben, so wird standardmäßig der Name a.out für das ausführbare Programm angenommen. Das Kommando cc liefert dem Linker die korrekten Argumente für C-Programme. Interessant wird die Linker-Option immer dann, wenn abweichend von einer gewöhnlichen Übersetzung mit Suchen von Funktionen in Standardbibliotheken eine Aktion erfolgen soll (siehe Beschreibung des Linkers).

G7.2. Überblick über die Programmiersprache C

Die Sprache C bietet:

- Typkonzept
- Blockstruktur
- maschinenunabhängiges Programmieren hardware-naher Operationen
- Makro-Preprozessor
- Standard-Include-Files
- Rekursive Prozeduren (call by value)
- separate Übersetzung
- Systemaufrufe.

Nicht unmittelbar in der Sprache enthalten sind:

- Operationen auf den zusammengesetzten Datentypen wie strings und arrays
- Ein-/Ausgaberoutinen
- mathematische Funktionen
- parallele Prozesse
- Prozeßsynchronisation
- zeitlich verzahnte Routinen.

C ist also eine "nackte" Sprache und all die fehlenden Ausdrucksmittel sind durch Bibliotheksfunktionen und Betriebssystemfunktionen bereitgestellt. Der Sprachumfang von C ist deshalb vergleichsweise klein und der Compiler kompakt und schnell. Ein C-Programm kann physisch aus einem oder mehreren Text-Files bestehen.
C-Programme setzen sich aus einer Sammlung von Definitionen von Variablen und Funktionen zusammen.
Die Funktionen sind die einzige Art von Unterprogrammen. Sie enthalten den die Algorithmen beschreibenden Programmtext. Argumente werden an Funktionen übergeben, indem die Werte der Argumente kopiert werden. Die aufgerufene Funktion ist nicht in der Lage, das Argument außerhalb der Funktion zu verändern (call by value). Dies stellt aber keinen Nachteil dar. Auf diese Weise können Hilfsvariable eingespart werden, die sonst bei der von anderen Programmiersprachen gewohnten Parameterübergabe über Adresse (call by reference) erforderlich sind.
Es ist möglich, in der Sprache C als Funktionsargumente auch Zeiger zu übergeben und somit eine Parameterübergabe über Adresse zu realisieren. Die Funktion kann in solch einem Fall das Objekt, auf das der Zeiger zeigt, verändern, wobei der Zeiger selbst wertmäßig gleich bleibt.

Jede Funktion kann rekursiv aufgerufen werden. Funktionen dürfen nicht verschachtelt definiert werden. Die Funktionen können untereinander Daten sowohl über ihre Argumente als auch über globale Variable austauschen.

Werte werden in der Sprache C durch die von den heute üblichen Rechnern zur Verfügung gestellten Datenobjekte Byte, Wort und Mehrfachwort implementiert. Sie lassen sich entsprechend ihren Eigenschaften zu bestimmten Wertmengen, den sogenannten Datentypen, zusammenfassen.
Es gibt folgende grundlegende Datentypen:

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

Die Zahlen können mit verschiedenen Genauigkeiten angegeben werden.
Beim Aufzählungstyp werden, wie der Name sagt, die Werte aufgezählt. Sie haben den Charakter von Konstanten.
Aus den einfachen Datentypen können Hierarchien von Datentypen konstruiert werden. Es gibt in der Sprache C folgende höhere Datentypen:

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

Jede Variable gehört zu einem bestimmten Datentyp. Diese Information wird bei der Definition bzw. der Deklaration einer Variablen festgelegt. Es gibt die Möglichkeit, Variable zu initialisieren.

Ein Feld ist eine zusammengesetzte Datenstruktur, deren Komponenten vom gleichen Datentyp sind. Der Zugriff zu den Komponenten erfolgt über Indizes. Felder sind vergleichbar mit ein- oder mehrdimensionalen Matrizen.

Ein Verbund faßt Daten verschiedener Datentypen zu einer Einheit zusammen. Der Zugriff zu den Komponenten erfolgt über den Namen einer Verbundvariablen und den Komponentennamen.

Der Datentyp Gemeinsamer Bereich ermöglicht, daß eine Variable zu verschiedenen Zeitpunkten Objekte verschiedenen Datentyps und Speicherplatzbedarfs enthalten kann. Der Compiler sorgt für die Größe und die Ausrichtung des erforderlichen Speicherplatzes.

Eine Variable vom Datentyp Zeiger enthält die Adresse einer anderen Variablen oder einer Funktion.

In der Sprache C müssen alle Variablen vor ihrer Benutzung definiert sein. Eine Variablendefinition besteht aus dem Namen des Datentyps und einer Liste von Variablen. Der Gültigkeitsbereich von Variablen kann mit Hilfe von Schlüsselworten programmiert werden. Variable können lokal für eine Funktion, global innerhalb eines Files und schließlich global für das gesamte C-Programm sein.
Für Konstanten können symbolische Namen festgelegt, und mit diesen Namen kann anstelle der Konstanten operiert werden.

Innerhalb der Programmiersprache C gibt es zur Strukturierung des Programmablaufes folgende grundlegende Ausdrucksmittel:

- Block bzw. Anweisungsverbund,
- Bedingte Anweisung (ifififif),
- Schleifen mit Abbruchtest am Anfang (while, forwhile, forwhile, for),
- Schleifen mit Abbruchtest am Ende (dododo),
- Mehrwegeauswahl (switchswitchswitch) und
- Sprunganweisung (goto, break, continuegoto, break, continuegoto, break, continue).

In der Programmiersprache C gibt es folgende Klassen von Operatoren:

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

Mit Hilfe der Operatoren werden die Daten verknüpft. Die Ergebnisse werden anderen Datenobjekten zugewiesen oder beeinflussen die Steuerung des Programmablaufs.
Bei der Verarbeitung von Daten verschiedener Datentypen in einem Ausdruck realisiert der C-Compiler teilweise die Datenumwandlung automatisch. Variable, die von einem zusammengesetzten Datentyp sind, können nur komponentenweise verarbeitet werden.

8. Bibliotheken

Im Betriebssystem MUTOS stehen dem Nutzer zwei Bibliotheken zur Verfügung, durch deren Benutzung er Zugriff zu sämtlichen 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 enthält die Eintrittspunkte zu den Systemgrundfunktionen (Systemrufe). Weiterhin sind in dieser Bibliothek eine ganze Reihe von Funktionen archiviert, die auf den Systemrufen aufbauen und mit mehr Komfort für den Anwender verschiedene Verbindungen zu Systeminformationen und -diensten realisieren sowie wirksame Hilfsmittel zur Durchführung der in Nutzerprogrammen vorgesehenen E/A-Arbeit bereitstellen. Kernstück der letztgenannten Bibliothek ist das Standard-E/A-Paket. Die darin enthaltenen Funktionen und Makros unterstützen die Grundfunktionen öffnen, Schließen, Löschen von Files, Speicherplatzzuweisung, Pufferbehandlung, Positionieren innerhalb von Files, Testen möglicher Fehler sowie Lesen und Schreiben.
Das Standard-E/A-Paket zeichnet sich durch Einfachheit in seiner Benutzung aus. Die darin enthaltenen Funktionen sind mit dem Ziel einer möglichst hohen Zeit- und Speicherplatzeffektivität geschrieben worden und garantieren durch ihre Maschinenunabhängigkeit die Portabilität 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 benötigten Funktionen bereitgestellt werden.

9. Generierung

MUTOS 1700 wird mit Hilfe einer Objektgenerierung aufgebaut. Außerdem sind die Quellen des Betriebssystemkerns im Lieferumfang enthalten. Sie werden jedoch für die Generierung nicht benötigt. Jeder Anwender kann sich aus den vorhandenen Moduln ein Betriebssystem entsprechend seinen Bedingungen aufbauen. Wesentlicher Bestandteil der Generierung ist das Festlegen der konkreten Gerätekonfiguration bezüglich Typ der Geräte, E/A-Adressen und Interrupt-Vektoren. Außerdem werden die Standardgeräte für das Root-File-System sowie die Swap- und Pipe-Bereiche festgelegt. Wenn die angeschlossenen Geräte die Standardadressen und Standard-Interrupt-Vektoren belegen, braucht in den Quellen des Kerns nichts geändert zu werden.
In einem File oder im Dialog wird die gewünschte Konfiguration definiert. Ein Generierungsprogramm (config) verarbeitet diese Information und baut die Belegung der Trap- und Interrupt-Vektorbereiche sowie einige interne Tabellen auf.
Zur Unterstützung der Generierung wird ein Steuer-File für das Dienstprogramm make bereitgestellt, so daß nur wenige Bedienerhandlungen erforderlich sind.
MUTOS wird mit einem Basissystem ausgeliefert, das für Standardkonfigurationen und mittlere Anforderungen vorgesehen ist. Bei abweichenden Forderungen kann sich der Anwender ohne größere Probleme ein spezielles System generieren. Die generierten Betriebssystemvarianten sind grundsätzlich externspeicherresident.

Sachwortverzeichnis
~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~